Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review

Henrik Jacobsson
henrik.jacobsson@his.se
School of Humanities and Informatics, University of Skövde, Skövde, Sweden, and Department of Computer Science, University of Sheffield, United Kingdom.

Rule extraction (RE) from recurrent neural networks (RNNs) refers to finding models of the underlying RNN, typically in the form of finite state machines, that mimic the network to a satisfactory degree while having the advantage of being more transparent. RE from RNNs can be argued to allow a deeper and more profound form of analysis of RNNs than other, more or less ad hoc methods. RE may give us understanding of RNNs in the intermediate levels between quite abstract theoretical knowledge of RNNs as a class of computing devices and quantitative performance evaluations of RNN instantiations. The development of techniques for extraction of rules from RNNs has been an active field since the early 1990s. This article reviews the progress of this development and analyzes it in detail. In order to structure the survey and evaluate the techniques, a taxonomy specifically designed for this purpose has been developed. Moreover, important open research issues are identified that, if addressed properly, possibly can give the field a significant push forward.

1 Introduction

In this review, techniques for extracting rules (or finite state machines) from discrete-time recurrent neural networks (DTRNNs, or simply RNNs) are reviewed. We propose a new taxonomy for classifying existing techniques, present the techniques, evaluate them, and produce a list of open research issues that need to be addressed.

By rule extraction from RNNs (hereafter denoted RNN-RE), we refer to the process of finding and building (preferably comprehensible) formal computational models and machines that mimic the RNN to a satisfactory degree. The connection between RNNs and formal models of computation is almost as old as the study of RNNs themselves, as the origins of these fields are largely overlapping. The study of neural networks once coincided with the study of computation in the binary recurrent network implementations of finite state automata of the theoretical work on nervous systems by McCulloch and Pitts (1943) (an interesting overview of this topic is found in Forcada, 2002). This common heritage has been flavoring the development

of the digital computer, although our current computer systems are far from being models of the nervous system.

In the early 1990s, the research on recurrent neural networks was revived. When Elman (1990) introduced his quite well-known simple recurrent network (SRN), the connection between finite state machines and neural networks was again there from the start. In his paper, he compared the internal activations of the networks to the states of a finite state machine.

In theory, RNNs are Turing equivalent (Siegelmann & Sontag, 1995) and can therefore compute whatever function any digital computer can compute.[1] But we also know that getting the RNN to perform the desired computations is very difficult (Bengio, Simard, & Frasconi, 1994). This leaves us in a form of knowledge vacuum; we know that RNNs can be immensely powerful computational devices, and we also know that finding the instantiations of RNNs that perform these computations could very well be an insurmountable obstacle, but we do not have the means for efficiently determining the computational abilities of our current RNN instantiations. On a less theoretical level, we can simply evaluate the performance of different RNNs to see to which extent we solved the learning problem for a specific domain. Such studies are conducted in virtually all papers applying RNNs on a domain, and in some cases more systematic studies are presented (Miller & Giles, 1993; Horne & Giles, 1995; Alquézar, Sanfeliu, & Sainz, 1997). But even something as simple as evaluating the performance of an RNN on a specific domain has some intrinsic problems since implicit aspects of the evaluation procedure can have a significant impact on the estimated quantitative performance (Jacobsson & Ziemke, 2003a).

Actually, the analysis problems may lead to the use of models that are too simplistic, such as smaller networks and toy problem domains, just to be able to analyze (or visualize) the results. One may wonder how many published networks with only two or three state (or hidden) nodes had their specific topology chosen just to make the plotting of their internal activations possible. What we need to do is in-depth analyses of RNN instantiations to uncover the actual behavior of RNN instantiations without the need for "manually" analyzing visualizations of the RNN behavior. An efficient rule extraction technique may be the best tool for such analyses.

1.1 Topic Delimitation. Since the early 1990s, an abundance of articles has been written on recurrent neural networks, and many of them have dealt explicitly with the connection between RNNs and state machines.[2] Many contributions have been theoretical, establishing the connection between (analog) RNNs (or other dynamical systems) and traditional (discrete)

[1] Actually McCulloch and Pitts (1943) had already determined this equivalence in 1943 for discrete networks (Medler, 1998).

[2] Many of these are summarized in Kremer (2001) and Barreto, Araújo, and Kremer (2003).

computational devices (Crutchfield & Young, 1990; Servan-Schreiber, Cleeremans, & McClelland, 1991; Crutchfield, 1994; Kolen, 1994; Horne & Hush, 1994; Siegelmann & Sontag, 1995; Casey, 1996; Tiňo, Horne, Giles, & Collingwood, 1998; Jagota, Plate, Shastri, & Sun, 1999; Omlin & Giles, 2000; Sima & Orponen, 2003; Hammer & Tiňo, 2003; Tiňo & Hammer, 2003). This work covers a wide spectrum of highly interesting and important theoretical insights, but in this article, we will not dwell on these theoretical issues because they are not the focus of this survey and because some of these articles are already much like surveys themselves, summarizing earlier findings.

On the pragmatic side, we find articles describing techniques for transforming state machines into RNNs (rule insertion) or for transforming RNNs into state machines (rule extraction) (e.g., Omlin & Giles, 1992, 1996a, 1996c, 2000; Giles & Omlin, 1993; Das, Giles, & Sun, 1993; Alquézar & Sanfeliu, 1994a; Omlin, Thornber, & Giles, 1998; Carrasco, Forcada, Muñoz, & Ñeco, 2000; Carrasco & Forcada, 2001). This article, however, deals exclusively with algorithms for performing rule extraction from RNNs.

Unfortunately, there will not be much room for discussion around analysis tools of RNNs other than just RE. There are a multitude of methods used to analyze RNNs, and a survey on this issue should definitely be written as well. A brief (and most probably inconclusive) list of examples of other analysis tools that have been used on RNNs is Hinton diagrams (e.g., Hinton, 1990; Niklasson & Bodén, 1997), hierarchical cluster analysis (e.g., Cleeremans, McClelland, & Servan-Schreiber, 1989; Elman, 1990; Servan-Schreiber, Cleeremans, & McClelland, 1989; Sharkey & Jackson, 1995; Bullinaria, 1997), simple state space plots (e.g., Giles & Omlin, 1993; Zeng, Goodman, & Smyth, 1993; Gori, Maggini, & Soda, 1994; Niklasson & Bodén, 1997; Tonkes, Blair, & Wiles, 1998; Tonkes & Wiles, 1999; Rodriguez, Wiles, & Elman, 1999; Rodriguez, 1999; Tabor & Tanenhaus, 1999), activation values plotted over time (e.g., Husbands, Harvey, & Cliff, 1995; Meeden, 1996; Ziemke & Thieme, 2002), iterated maps (e.g., Wiles & Elman, 1995), vector flow fields (e.g., Rodriguez et al., 1999; Rodriguez, 1999), external behavior analysis of RNN-controlled autonomous robotic controllers (e.g., Husbands et al., 1995; Meeden, 1996), weight space analysis (e.g., Bodén, Wiles, Tonkes, & Blair, 1999; Tonkes & Wiles, 1999), dynamical systems theory (e.g., Tonkes et al., 1998; Rodriguez et al., 1999; Rodriguez, 1999; Bodén, Jacobsson, & Ziemke, 2000), and ordinary quantitative evaluations of RNN performance for different domains (basically every article where an RNN is applied).

Unlike previous surveys on rule extraction (Andrews, Diederich, & Tickle, 1995; Tickle, Andrews, Golea, & Diederich, 1997, 1998), this article deals exclusively with rule extraction from recurrent neural networks (resulting in quite different evaluation criteria than in previous RE surveys, as you will see in section 3). In fact, many of the RE approaches for nonrecurrent networks could potentially be used on RNNs, or at least on nonrecurrent networks in temporal domains (e.g., Craven & Shavlik, 1996;

Sun, Peterson, & Sessions, 2001). There are also other symbolic learning techniques for "training" finite automata on symbolic sequence domains directly, without taking the extra step of training a neural network, that could be mentioned (Sun & Giles, 2001; Cicchello & Kremer, 2003). These techniques are certainly interesting in themselves and should also be compared to RNN-RE techniques experimentally, but this is something that is not further examined in this review.

To summarize, this review is oriented solely around RNN-RE techniques, but this field is closely related to the areas already mentioned. It may also be worth mentioning that as a review of techniques, this review is not a tutorial. Readers interested in replicating the techniques should refer to the cited articles.

1.2 Contents Overview. In section 2 we describe RNNs and finite state machines and common characteristics of RNN-RE algorithms. The evaluation criteria underlying the construction of a taxonomy for appropriately classifying and describing RNN-RE algorithms are described in section 3. The techniques are then described in section 4, and important criticism of RNN-RE in general is discussed in section 5. The set of existing RNN-RE techniques is discussed in the light of the evaluation criteria in section 6, and open research issues are summarized in section 7. In section 8, we present some conclusions.

2 Background

An RNN processes sequences of data (input) and generates a responses (output) in a discrete time manner. The RNN processes information by using its internal continuous state space as an implicit, holistic memory of past input patterns (Elman, 1990). In the extraction of rules from an RNN, the continuous state space is approximated by a finite set of states, and the dynamics of the RNN is mapped to transitions among this discrete set of states.

We now give a brief definition of what constitutes a recurrent neural network in the scope of this review. A brief introduction to finite state machines (FSMs) is also given, since the extracted rules are typically represented as such. A more detailed description of what RNN-RE algorithms typically constitute then follows.

2.1 Recurrent Neural Networks. A detailed review of the achievements in RNN research and the vast variety of different RNN architectures is far beyond the scope of this article. Instead, a set of identified common features of most RNN architectures will be described at an abstract enough level to incorporate most networks to which the existing RNN-RE algorithms could be applied and also abstract enough to see the striking similarities of RNN computation with the computation in

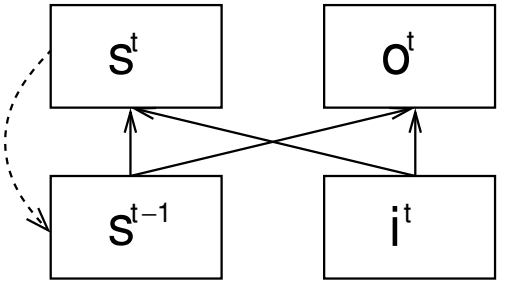

Figure 1: Functional dependencies of the input, state, and output of an RNN.

finite state machines (see section 2.2). Readers with no prior experience of RNNs can find more detailed description and well-developed classifications of RNNs in Kolen and Kremer (2001), Kremer (2001), or Barreto et al. (2003).

Only a few of the many RNN architectures have been used in the context of rule extraction; among them are simple recurrent networks (SRNs; Elman, 1990) and, more commonly, second-order networks (e.g., sequential cascaded Networks, SCNs; Pollack, 1987). These models differ somewhat in their functionality and how they are trained. But the functional dependencies are at some level of abstraction basically the same, which is going to be exploited in the definition below.

A recurrent neural network R is a six-tuple $R = \langle I, O, S, \delta, \gamma, \mathbf{s}^0 \rangle$, where $I \subseteq \mathbb{R}^{n_i}$ is a set of input vectors, $S \subseteq \mathbb{R}^{n_s}$ is a set of state vectors, $O \subseteq \mathbb{R}^{n_o}$ is a set of output vectors, $\delta : S \times I \to S$ is the state transition function, $\gamma : S \times I \to O$ is the state interpretation function, and $\mathbf{s}^0 \in S$ is the initial state vector. $n_i, n_s, n_o \in \mathbb{N}$ are the dimensionalities of the input, state, and output spaces, respectively.

Often (or perhaps always) the input, state, and output are restricted to hypercubes with all elements limited to real numbers (or rational approximations of real numbers when simulated) between 0 and 1 or -1 and 1. When the networks are being trained, the two functions δ and γ are typically adjusted to produce the desired output according to some training set. For a sequence of input vectors $(\mathbf{i}^1, \mathbf{i}^2, \ldots, \mathbf{i}^\ell)$, the state is updated according to $\mathbf{s}^t = \delta(\mathbf{s}^{t-1}, \mathbf{i}^t)$ and the output according to $\mathbf{o}^t = \gamma(\mathbf{i}^t, \mathbf{s}^{t-1})$. The functional dependencies are depicted in Figure 1.

Note that the weights, biases, activation functions, and other concepts we typically associate with neural networks are all hidden in the state transition function δ and state interpretation function γ. The reason is that as far as RNN-RE algorithms are concerned, the fact that the networks have adaptive weights and can be trained is of less importance. An interesting consequence of the abstract nature of this RNN description,

and that this is all that is required to continue describing RNN-RE algorithms, is that it tells something about the portability of the algorithms (cf. section 3.2). There are simply not many assumptions and requirements of the underlying RNNs, which means that they are portable to more RNN types than they would be otherwise. There are a few assumptions, though—for example, that states should cluster in the state space as a result of the training (Cleeremans et al., 1989; Servan-Schreiber et al., 1989). Some, more implicit, assumptions are also the target for some of the criticism of RE from RNNs (Kolen 1993, 1994), which will be discussed in section 5 (more implicit assumptions are also discussed in section 6.5).

2.2 Finite State Machines. The rules extracted from RNNs are almost exclusively represented as FSMs. The description here will be kept brief. For a full discussion of what a regular language is and what other classes of languages there are, interested readers are referred to Hopcroft and Ullman (1979).

A deterministic Mealy machine M is a six-tuple $M = \langle X, Y, Q, \delta, \gamma, q^0 \rangle$, where X is the finite input alphabet, Y is the finite output alphabet, Q is a finite set of states, $\delta : Q \times X \to Q$ is the transition function, $\gamma : Q \times X \to Y$ is the output function, and $q^0 \in Q$ is the initial state (note the similarities with the definition of RNNs in section 2.1).

In cases where the output alphabet is binary, the machine is often referred to as a finite state automation (FSA). In an FSA, the output is interpreted as an accept or reject decision determining whether an input sequence is accepted as a grammatical string.

There are actually two different models for how an FSM can be described—Mealy (as above) or Moore machines; although they are quite different from each other, they are computationally equivalent (Hopcroft & Ullman, 1979). Moore machines generate outputs based on only the current state and Mealy machines on the transitions between states; the output function, γ, is for a Moore machine $\gamma : Q \to Y$ and for a Mealy machine $\gamma : Q \times X \to Y$.

In deterministic machines, an input symbol may trigger only a single transition from one state to exactly one state (as in the definition above). In a nondeterministic machine, however, a state may have zero, one, or more outgoing transitions triggered by the same input, that is, the transition function, δ, is $\delta : Q \times X \to 2^Q$ (a function to the power set of Q) instead of $\delta : Q \times X \to Q$. That means that in a nondeterministic machine, a symbol may trigger one or more transitions from a state or even no transition at all (since $\emptyset \in 2^Q$). We will denote nondeterministic machines incomplete if there is at least one $q \in Q$ and $x \in X$ such that $\delta(q, x) = \emptyset$. Deterministic and nondeterministic machines are computationally equivalent, although nondeterministic machines can typically be much more compact (i.e., have

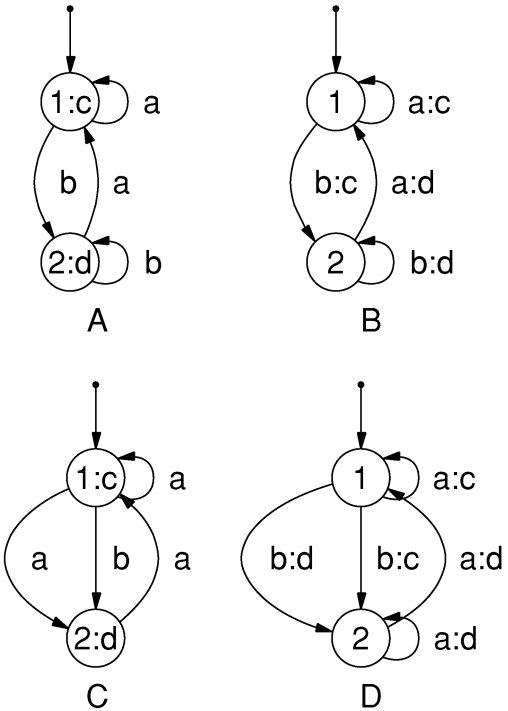

Figure 2: Examples of (nonequivalent) different FSM types with $X = \{a, b\}$, $Y = \{c, d\}$, $Q = \{1, 2\}$, and $q_o = 1$. (A) Deterministic Moore machine. (B) Deterministic Mealy machine. (C) Nondeterministic Moore machine. (D) Nondeterministic Mealy machine.

fewer states) than their deterministic counterpart. Deterministic FSM, and deterministic FSAs will be abbreviated DFM and DFA, respectively.

In summary, there are four types of FSMs: deterministic Moore machine, deterministic Mealy machine, nondeterministic Moore machine, and nondeterministic Mealy machine (see Figure 2 for examples). Moreover, the machines can be stochastic as well if transition probabilities are also encoded in the machine.[3]

For a more detailed description of deterministic and nondeterministic, Mealy and Moore machines, proof of equivalence, and a "standard" minimization algorithm, see Hopcroft and Ullman (1979). For the corresponding theory on stochastic machines, see Paz (1971).

[3] Cf. Stochastic sequential machines (Paz, 1971) and probabilistic automata (Rabin, 1963).

Table 1: The Common Ingredients of RNN-RE Algorithms.

1. Quantization of the continuous state space of the RNN, resulting in a discrete set of states
2. State and output generation (and output classification, if necessary) by feeding the RNN input patterns
3. Rule construction based on the observed state transitions
4. Rule set minimization

2.3 The Basic Recipe for RNN Rule Extraction. The algorithms described in this article have many features in common (see Table 1).

The continuous state space of the RNN needs to be mapped into a finite set of discrete states corresponding to the states of the resulting machine. We will refer to the states of the network as *microstates* and the finite set of quantized states of the network as *macrostates*. The macrostates are basically what the RE algorithm "sees" of the underlying RNN, whereas the actual state of the network, the microstates, is hidden. The act of transforming the microstates into macrostates is a critical part of RNN-RE algorithms (ingredient 1 in Table 1) and is called *quantization*. One macrostate corresponds to an uncountable set of possible microstates (just in theory; in practice, the RNN is simulated on a computer with finite precision). Therefore, deterministic sequences of events at the microstate level may appear stochastic at the macrostate level since information is lost in the quantization; for example, if two microstates $a_1, a_2 \in A$ deterministically transit to microstates $b_1 \in B$ and $c_1 \in C$, respectively, then at the macrostate level, it cannot be determined from observing macrostate A whether the next macrostate will be B or C.

Another common ingredient of RNN-RE algorithms is systematic testing of the RNN with different inputs (from the domain or generated specifically for the extraction) and the (macro)states and outputs are stored and used to induce the FSM (ingredient 2 in Table 1). The third ingredient is the inherent machine construction, a process often conducted concurrently with the state and output generation.

Many times the generated machine is then minimized using a standard minimization algorithm (Hopcroft & Ullman, 1979); this is the fourth common ingredient of RNN-RE algorithms. FSM minimization is, however, not part of all algorithms and can also be considered as an external feature, independent of the actual extraction.

3 Evaluation Criteria and Taxonomy

To simplify comparisons and structure the descriptions of the algorithms, several evaluation criteria have been chosen. These criteria are used in the

tables containing summaries in section 4 and are of central importance to the discussions in section 6.

The rule type, quantization method, and state generation method can be considered to constitute the main distinguishing features of RNN-RE algorithms, and they have therefore been used to structure this survey.

3.1 Main Criteria

3.1.1 Rule Type. The rules generated by RNN-RE algorithms are FSMs that are either deterministic, nondeterministic, or stochastic. They can also be in a Mealy or Moore format. In our classification of rule types, we will also distinguish whether the machine (and underlying RNN) is producing a binary accept-reject decision at the end of a string (i.e., like an FSA) or if the task is to produce an output sequence of symbols based on the input sequence (typically for prediction).

3.1.2 Quantization. One of the most varying elements of existing RNN-RE algorithms is the state space quantization method. Examples of methods used are hierarchical clustering, vector quantization, and self-organizing maps (see section 6.2 for a detailed discussion).

3.1.3 State Generation. Another important criterion is the state generation procedure, for which there are two basic methods: searching and sampling. These will be described further in the descriptions of the algorithms.

3.1.4 Network Type and Domain. Although not a feature of the extraction algorithm per se, the network types and in which domains each RNN-RE algorithm has been used will be explicitly listed for each presented technique.

3.2 Criteria from the ADT Taxonomy. Andrews et al. (1995) introduced a taxonomy, the *ADT* taxonomy,[4] for RE algorithms, which has since been an important framework when introducing new or discussing existing RE algorithms (e.g., Schellhammer, Diederich, Towsey, & Brugman, 1998; Vahed & Omlin, 1999; Craven & Shavlik, 1999; Blanco, Delgado, & Pegalajar, 2000). The five evaluation criteria in the ADT taxonomy were expressive power, translucency, portability, rule quality, and algorithmic complexity. However, for some of their classification aspects, all RNN-RE algorithms would end up in the same class, and those aspects are therefore not very informative. The ADT taxonomy does, however, provide some very useful viewpoints taken in section 6. Some of the terminology from the

[4] ADT comes from the names of the authors: Andrews, Diederich, and Tickle.

ADT taxonomy will also appear in various places in this survey; therefore, a brief description of the ADT aspects is given below.

3.2.1 Expressive Power. The expressive power is basically the type of rules generated by the RE and hence subsumed by our rule type criteria. ADT identified (when also taking Tickle et al., 1997, 1998, into account) four basic classes:

- Propositional logic (i.e., *if . . . then . . . else*)
- Nonconventional logic (e.g., fuzzy logic)
- First-order logic (i.e., rules with quantifiers and variables)
- Finite state machines

Almost all rules from RNN-RE algorithms would fall into the last category.

3.2.2 Translucency. One of the central aspects in the ADT taxonomy, translucency, that is, the degree to which the rule-extraction algorithm "looks inside" the ANN, is less relevant here since it is not a distinguishing feature of RNN-RE algorithms. ADT initially identified three types of RE algorithms: (1) decompositional algorithms, where rules are built on the level of individual neurons and then combined; (2) pedagogical approaches using a black box model of the underlying network; and (3) eclectic algorithms with aspects from both previous types. Tickle et al. (1998) also introduced a fourth intermediate category, compositional, to accommodate RNN-RE algorithms that are all (except for one pedagogical algorithm (Vahed & Omlin, 1999, 2004)) based on analyzing ensembles of neurons (i.e., the hidden state space).

3.2.3 Portability. Portability describes how well an RE technique covers the set of available ANN architectures. As for translucency, the portability is probably much the same for all RNN-RE algorithms. It is also a quite complex aspect of RE techniques (tightly bound with translucency and, in terms of feasibility, with algorithmic complexity), and we have therefore chosen not to distinguish RNN-RE algorithms by this criterion.

3.2.4 Quality. The quality of the extracted rules is a very important aspect of RE techniques, and perhaps the most interesting for evaluation of the quality of the algorithms. This aspect differs from the others in that it evaluates RE algorithms at the level of the rules rather than the level of the RE algorithms themselves.

Based on previous work, such as Towell and Shavlik (1993), four subaspects of rule quality were suggested in the ADT taxonomy:

- *Rule accuracy*—the ability of the rules to generalize to unseen examples
- *Rule fidelity*—how well the rules mimic the behavior of the RNN

- *Rule consistency*—the extent to which equivalent rules are extracted from different networks trained on the same task
- *Rule comprehensibility*—the readability of rules or the size of the rule set

3.2.5 Algorithmic Complexity. The algorithmic complexity of RE algorithms is unfortunately also often an open question as authors seldomly analyze this explicitly (Andrews et al., 1995). Although Golea (1996) showed that RE can be an NP-hard problem, it is unclear how existing heuristics affect the actual expected time and space requirements. For RNN-RE, the complexity issue has not received much attention, and the issue itself is quite complex, as the execution time can be affected by many factors, such as the number of state nodes, number of input symbols, granularity of the quantization, and RNN dynamics.

4 RNN-RE Techniques

Although we have identified some common characteristics among the RE algorithms, dividing them into groups has been a painstaking task, as there are innumerable ways to do so. The techniques will be presented in a primarily chronological order. When a later technique is similar to an earlier one, it will be presented in connection with its predecessor (although this relation may be constituted by coincidental similarities rather than a direct continuation of prior work).

First, some early work that laid the ground for RE techniques to be developed will be presented. Then the algorithms will be described in more detail in sections 4.2 to 4.7. For fuller descriptions of the algorithms, we refer to the original articles.

4.1 Pre-RE Approaches. To understand the roots of FSM extraction from recurrent networks, it is useful to recognize that in some early attempts to analyze RNNs, clustering techniques were used on the state space, and clusters corresponding to the states of the FSM generating the language were found (clustering remains one of the central issues of the research on RE from RNNs). Hierarchical cluster analysis (HCA) was used for analyzing RNNs in a few early articles on RNNs (Cleeremans et al., 1989; Servan-Schreiber et al., 1989, 1991; Elman, 1990). The authors found that for a network trained on strings generated by a small FSM, the HCA may find clusters in the state space apparently corresponding to the states of the grammar. The clusters of the HCA were labeled using the labels of the states of the underlying (known) state machine, making it easy to draw the connection between the RNN and the FSM.

The fact that much of the early research on RNNs was conducted on problem sets explicitly based on FSMs may have biased subsequent research to

look for these FSMs inside the network. However, for some successful networks (e.g., Servan-Schreiber et al., 1991), no clusters corresponding directly to the states of the FSM, which generated the training set language, were found. This meant that the network had an alternative, but apparently correct, representation of the problem that differed from the one anticipated. This was probably due to the fact that the clusters of the internal state of the network did not necessarily have a straightforward one-to-one relation with the states of the corresponding minimal machine. It was later shown that nonminimal machines would typically be what is initially extracted by clustering the state space when RE was used on RNNs (Giles, Miller, Chen, Chen, & Sun, 1992). Therefore, FSM minimization is included in most RNN-RE algorithms.

The basic problem of using only clustering (and not recording the transitions) for analyzing RNNs is that there is no reliable way of telling how the clusters relate to each other temporally.[5] If the exact same FSM is not found, the clusters may not be labeled using the original FSM as a source and the temporal ordering of the clusters is therefore lost. This problem was also observed by Elman (1990): "The temporal relationship between states is lost. One would like to know what the trajectories between states . . . look like." The solution of this problem led to the development of FSM extraction from RNNs.

4.2 Search in Equipartitioned State Space. The algorithm of Giles et al. (Giles et al., 1991; Giles, Miller, Chen, Chen, et al., 1992; Omlin & Giles, 1996b) partitioned the state space into equally sized hypercubes (i.e., macrostates) and conducted a breadth-first search by feeding the network input patterns until no new partitions were visited. The transitions among the macrostates (induced by input patterns) were the basis for the extracted machine. The search started with a predefined initial state of the network and tested all possible input patterns on this microstate (see Figures 3 and 4). The first encountered microstate of each macrostate was then used to induce new states. This guaranteed the extraction of a deterministic machine since any state drift (Das & Mozer 1994, 1998) was avoided as the search was pruned when reentering already visited partitions. The extracted automaton was then minimized using a standard minimization algorithm for DFAs (Hopcroft & Ullman 1979). The algorithm is summarized in Table 2.

The central parameter of the algorithm is the quantization degree q of the equipartition. The authors suggested starting with $q = 2$ and increasing it until an automata consistent with the training set is extracted, that is, the

[5] There are other, more general problems of an HCA-based analysis of ANNs in general, as adjacent states (i.e., hidden unit activations) may be interpreted differently by the output layer and remote states may have the same interpretation (Sharkey & Jackson, 1995). For RNNs, this becomes even more problematic as the state is not only mapped into an output but also mapped recursively to all succeeding outputs through the state transitions.

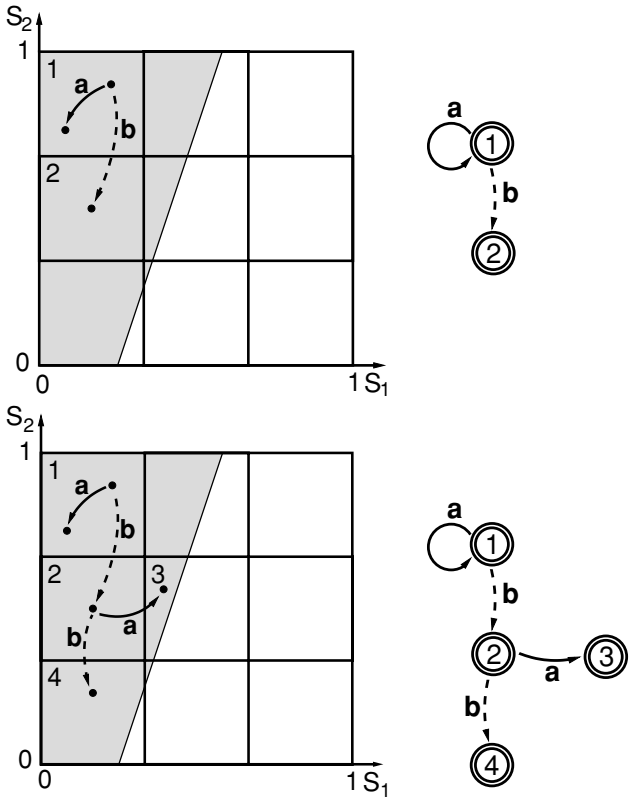

Figure 3: The two first iterations in an example of the DFA extraction algorithm of Giles et al. (1991) used on an RNN with two state nodes trained on a binary language and the quantization parameter $q = 3$. The state space is divided into an accept and reject region (gray and white, respectively). The algorithm expands the graph until all nodes have two outgoing arcs.

termination criteria is to have perfect accuracy of the rules. The choice of q is however usually not explicitly described as part of the RE algorithm (one exception is in the description by Omlin, 2001, where the suggested incremental procedure is also part of the algorithm).

Giles, Miller, Chen, Chen, et al. (1992) found that the generalization ability of the extracted machines sometimes exceeded that of the underlying RNNs. Since the networks were trained on regular grammars, if the extraction result was a DFA equivalent with the original grammar that generated the training-test set, generalization would also be perfect. Giles, Miller, Chen, Sun, et al. (1992) showed that during successful training of an RNN, the extracted DFA will eventually belong to the same equivalence class as the

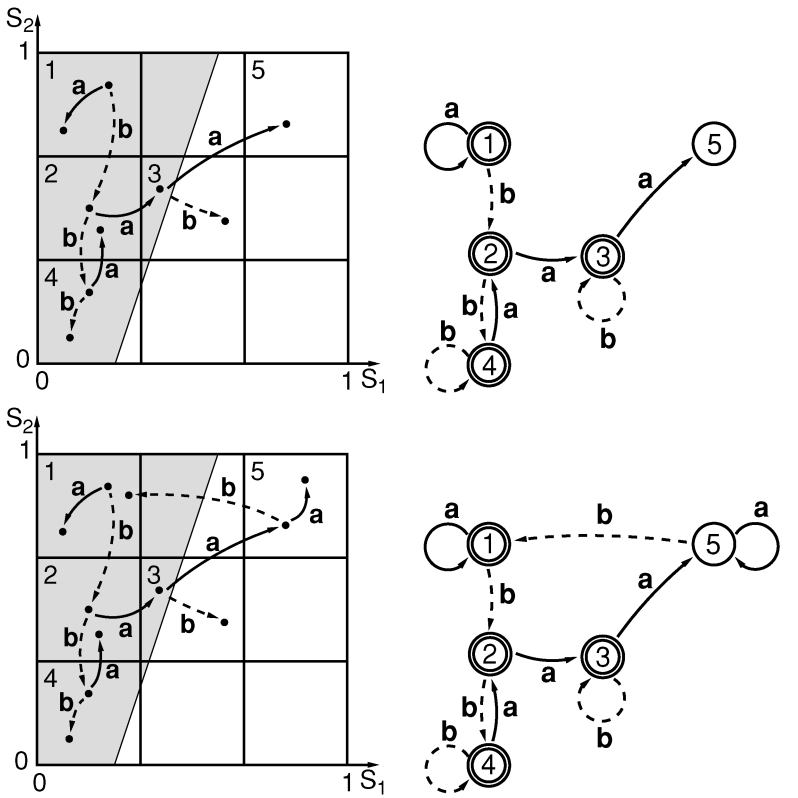

Figure 4: The two final iterations in the example of the DFA extraction algorithm of Giles et al. (1991). Note that the macrostate corresponding to node 3 could be interpreted as both an accept *and* reject state depending on the microstate, but the algorithm used the interpretation of the first encountered microstate as the interpretation of the macrostate.

original DFA. Existence of equivalence classes over different degrees of quantization (i.e., different values of q) was used in, Omlin, Giles, & Miller (1992) as an indicator of the networks' generalization ability; if the extracted DFAs for increasing values of q collapsed into a single equivalence class, it was taken as a sign of good generalization ability without the need for explicitly testing this on a separate test set.

The same algorithm has been used in various other contexts: as part of rule refinement techniques (e.g., Omlin & Giles, 1992, 1996c; Giles & Omlin, 1993; Das et al., 1993), as an indicator of underlying language class (Blair & Pollack, 1997), as a method for complexity evaluation (e.g., Bakker & de Jong, 2000), as part of a quantitative comparison of different RNN architectures

Table 2: Summary of Algorithm Extracting DFA Through Searching in an Equipartitioned State Space.

DFA Extraction, Regular Partitioning, Breadth-First Search (Giles et al., 1991; Giles, Miller, Chen, Chen, et al., 1992; Omlin & Giles, 1996b)	
Rule type	Moore DFA with binary (accept-reject) output
Quantization	Regular partitioning by q intervals in each state dimension, generating q^N bins of which typically only a small subset is visited by the RNN
State generation	Breadth-first search
Network(s)	Predominantly used on second-order RNNs
Domain(s)	Predominantly regular languages with relatively few symbols; some applied domains, e.g., quantized financial data (Giles, Lawrence, & Tsoi 1997, 2001; Lawrence, Giles, & Tsoi, 1998)

(Miller & Giles, 1993), as a means for FSM acquisition (e.g., Giles, Horne, & Lin 1995),[6] or simply as an analysis tool of the RNN solutions (e.g., Giles & Omlin, 1994; Goudreau & Giles, 1995; Giles et al., 1997; Lawrence et al., 1998; Lawrence, Giles, & Fong, 2000; Giles et al., 2001; Bakker, 2004).[7] The algorithm has also been used in the context of recursive networks (Maggini, 1998).

An apparent problem with this technique is that the worst-case number of clusters grows exponentially with the number of state nodes N (q^N). The time needed for the breadth-first search will also grow exponentially with the number of possible input symbols. In practice, however, the number of visited states is much smaller than the number of possible states.

This, the earliest of RNN-RE methods, is also the most widely spread algorithm. Almost all following articles where new RNN-RE techniques have been proposed cite Giles, Miller, Chen, Chen, et al. (1992). But often these articles do not contain citations to each other, giving the first impression of the field as less diverse than it actually is. Consequently, there is a surprising variety of RE approaches, some of them seemingly developed independently of each other.

4.3 Search in State Space Partitioned Through Vector Quantization. An alternative to the simple equipartition quantization was suggested by Zeng et al. (1993) where a k-means algorithm was used to cluster the microstates. The centers of the clusters, the model vectors, were used as the basis for the breadth-first search; the RNN was tested with all input symbols for each model vector state (cf. the equipartition algorithms, where the

[6] Implicitly, however, more or less all articles using RE are in some way on FSM/language acquisition.

[7] This is also implicitly part of many other articles as well.

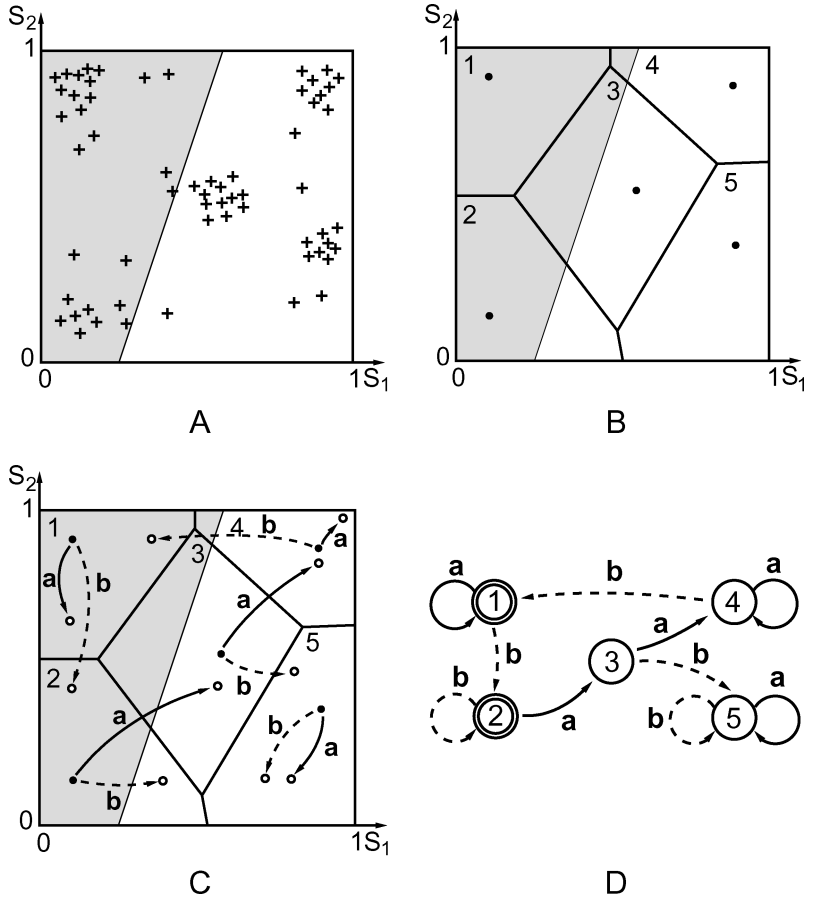

Figure 5: An illustrative example of rule extraction through breadth-first search in a state space clustered by k-means. (A) The states of the RNN are sampled during training. (B) These states are clustered into a predefined number of clusters. (C) A breadth-first search (cf. Figures 3 and 4) is conducted based on the model vectors. (D) The machine is constructed.

first encountered RNN state is the basis for further search). (See Figure 5 for an illustrative example of this algorithm.) A similar approach, also using k-means, developed seemingly independently from Zeng et al. (1993), was presented in Frasconi, Gori, Maggini, and Soda (1996) and Gori, Maggini, Martinelli, and Soda, 1998, and a similar SOM-based approach in Blanco et al. (2000). A summary of these approaches is given in Table 3.

To support an appropriate clustering of states, Zeng et al. (1993) and Frasconi et al. (1996) induced a bias for the RNN to form clusters during

Table 3: Summary of Algorithms Extracting DFA Through Searching in a State Space Partitioned by Vector Quantization.

DFA Extraction, Vector quantifier, Breadth-First Search (Zeng et al., 1993; Frasconi et al., 1996; Gori et al., 1998)	
Rule type	Moore DFA with binary (accept-reject) output
Quantization	k-means
State generation	Breadth-first search
Network(s)	Second-order RNNs (Zeng et al., 1993), recurrent radial basis function network (Frasconi et al., 1996; Gori et al., 1998), RNN with an external pushdown automaton (Sun, Giles, &, Chen, 1998)
Domain(s)	Regular binary languages (Tomita, 1982), context free languages (Sun et al., 1998)

Table 4: Summary of the Search-Based DFA Extracting Algorithm Proposed by Alquézar and Sanfeliu for Unbiased Grammars.

DFA Extraction, Hierarchical Clustering, Sampling on Domain (Alquézar & Sanfeliu, 1994a; Sanfeliu & Alquézar, 1995)	
Rule type	Unbiased Moore DFA; unbiased means the output is trinary (accept, reject, and unknown)
Quantization	Hierarchical clustering
State generation	A prefix tree is built based on the examples of the training set
Networks	First-order RNN (not specified in Alquézar and Sanfeliu, 1994a, but in Sanfeliu & Alquézar, 1995)
Domains	At least 15 different regular binary languages (Alquézar et al., 1997)

training. Other studies have also followed this approach (Das & Das, 1991; Das & Mozer, 1994, 1998). RE-RNN algorithms developed on such specialized RNNs may not work on other networks, however. RE techniques that can be used on already existing networks (i.e., typically not designed to be easy to analyze) are described by Tickle et al. (1998) as more attractive techniques.

In the presented search-based approaches, the reentering into partitions was the basis of pruning the search. A different pruning strategy was suggested by Alquézar and Sanfeliu, (1994a) and Sanfeliu and Alquézar (1995), who chose to use the domain to determine search depth (the algorithm is summarized in Table 4). A prefix tree (see Figure 6) was built based on the occurrences of positive and negative strings in the training set; the prefix tree contained only strings present in the training set. The states of the RNN were generated using only the strings in the prefix tree. The authors used RE

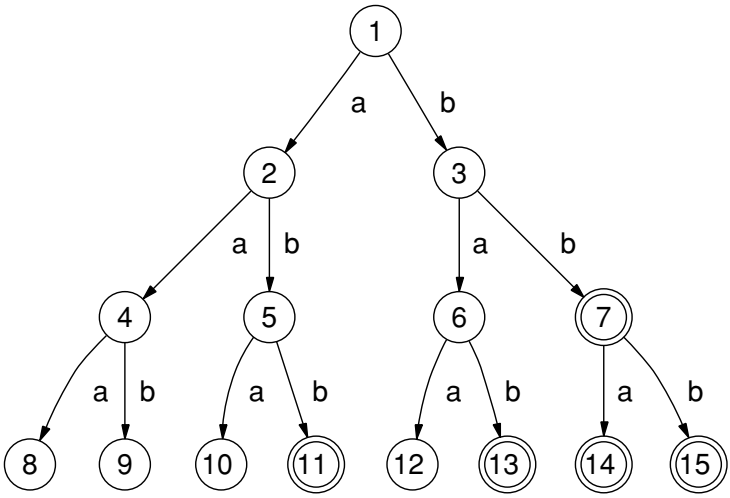

Figure 6: An example of a prefix tree of depth 3, created from a language that accepts only strings containing at least two b's.

as part of their active grammatical inference (AGI) learning methodology, an iterative rule refinement technique.

The states generated with the prefix tree were the basis of the initial machine. The spatially closest pair of these states was then merged iteratively until further clustering would result in an inconsistency. This RE technique was also used for a wide variety of regular grammars and two types of networks in Alquézar et al. (1997). The authors reported that the extracted machines on average performed significantly better than the original RNNs.

4.4 Sampling-Based Extraction of DFA. Instead of conducting a search in the quantized state space, activity of the RNN in interaction with the data and environment can be recorded. In this way, the domain can be considered as heuristics confining the states of the RNN to relevant states.

Before RE techniques for RNNs were developed, sampling of the state space using the domain was the most natural way to conduct analysis of RNNs (Cleeremans et al., 1989; Servan-Schreiber et al., 1989; Elman, 1990). The first RE technique based on sampling the RNN was proposed by Watrous and Kuhn (1992) (see Table 5). The quantization of the state space was based on splitting individual state units' activations into intervals. They noted that these intervals could be merged and split to help the extraction of minimal and deterministic rules. The procedure of state splitting, however, is a bit vaguely described and may require intervention from the user.

Manolios and Fanelli (1994) chose to use a simple vector quantifier to discretize the state space. Training from different, randomly initiated, model

Table 5: Summary of the Sampling-Based DFA Extraction Algorithm Proposed by Watrous and Kuhn (1992).

DFA Extraction, Dynamic Interval Clustering, Sampling on Domain (Watrous & Kuhn, 1992)	
Rule type	Moore DFA with binary (accept-reject) decision
Quantization	Dynamically updated intervals for each state unit; states are collapsed and split through updating the intervals
State generation	Sampling the RNN while processing the domain
Networks	Second-order RNNs
Domains	Regular binary languages (Tomita, 1982)

Table 6: Sampling-Based DFA Extractor Proposed Originally in Fanelli (1993).

DFA Extraction, Vector Quantifier, Sampling on Domain (Manolios & Fanelli, 1994; originally in Fanelli, 1993)	
Rule type	Moore DFA with binary (accept-reject) decision
Quantization	A simple vector quantifier; details unclear
State generation	Sampling on a test set
Networks	First-order RNNs
Domains	Regular binary languages (Tomita, 1982)

vectors were repeatedly conducted until a deterministic machine was found. The termination of this procedure is however not guaranteed. The algorithm is summarized in Table 6.

A similar approach was suggested in Tiňo and Šajda (1995), where an algorithm for removing inconsistent transitions was introduced. This algorithm could, however, fail under certain circumstances, so that the extraction of a DFA could not be guaranteed. A star topology self-organizing map (SOM; Kohonen, 1995) was used to quantize the state space. Tiňo and Šajda (1995) were the first to extract Mealy instead of Moore machines and also the first who did not confine the output to binary accept-reject decisions (not counting the unbiased DFA of Alquézar & Sanfeliu, 1994a). This algorithm is summarized in Table 7.

The breadth-first search will reliably find consistent DFMs since the search is pruned before inconsistencies leading to indeterminism are introduced. The DFM will also be complete since all symbols are tested on all states. In sampling the state space, determinism is no longer guaranteed since two microstates of the same macrostate may result in transitions to different macrostates, triggered by the same symbol. Two state vectors in the same partition may also be mapped to different classes in the output. The extracted machines may also be incomplete since not all symbols may be tested on all states. Therefore, the DFM extraction through sampling may fail, as in the above cases of Watrous and Kuhn (1992), Manolios and Fanelli (1994), and Tiňo and Šajda (1995). It is unclear how incomplete machines

Table 7: Summary of the Sampling-Based DFM Extractor of Tiňo and Šajda (1995).

DFM Extraction, SOM, Sampling on Domain (Tiňo & Šajda, 1995)	
Rule type	Mealy DFM with multiple output symbols
Quantization	Star topology SOM
State generation	Sampling on training set
Networks	Second-order RNNs
Domains	Regular formal language domains with either two or three input symbols (not counting the end-of-string symbol) and two or three output symbols

Table 8: Summary of the Only Sampling-Based DFM Extractor Where Inconsistencies and Incompleteness Are Handled.

DFM Extraction, Vector Quantizer, Sampling on Domain (Schellhammer et al., 1998)	
Rule type	Mealy DFM with a "rescue state" used to make machine complete
Quantization	k-means
State generation	Sampling on training set; inconsistencies solved by discarding the least frequent of inconsistent transitions
Networks	SRN
Domains	Natural language prediction task

were handled in the described approaches; perhaps the extracted machines were small enough and domains simple enough that such problems did not arise.

An approach to solve the problem of indeterminism is to use transition frequencies to discard the least frequent of inconsistent transitions. This heuristic should in most cases solve the inconsistency without deviating much from the operation of the underlying RNN in the majority of the transitions. This simple procedure was proposed by Schellhammer et al. (1998) (summarized in Table 8). They also handled the problem of incomplete machines by creating transitions to a predefined "rescue state" to make the machine complete. These simplifications did not significantly reduce the performance of the DFM, and the rescue state made it possible for the machines to make "guesses" about inputs that otherwise would not be possible to parse.

4.5 Stochastic Machine Extraction. The extraction of deterministic FSMs (DFMs) from RNNs through sampling is hampered by the fact that the quantization of the state space may lead to inconsistencies in the macrostate transitions. These inconsistent transitions (and potentially state

Table 9: Summary of Approaches of RNN-RE for Extraction of Stochastic Machines.

Stochastic Machine Extraction, SOM, Sampling on Domain (Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999)	
Rule type	Stochastic Mealy finite state machine
Quantization	SOM (unspecified topology) in Tiňo and Vojtek (1998) and DCS in Tiňo and Köteles (1999)
State generation	Two phases: Sampling on training set and self-driven RNN
Networks	Primarily second-order RNNs.
Domains	Prediction of (four) symbols generated from continuous chaotic laser data and a chaotic series of binary symbols generated with iterated logistic map function

interpretations) will, however, follow some patterns, and if all such transitions are counted, they can be transcribed into a stochastic machine—a machine with probabilities associated with the transitions. The inconsistencies that ruin a DFM extraction may, in other words, contain informative probabilities that more accurately describe the RNN.

An algorithm for extraction of stochastic machines from RNNs was proposed by Tiňo and Vojtek (1998). These extracted machines were, however, not equivalent to the stochastic machines as defined by Paz (1971) and Rabin (1963) in that the conditional probability of the output given the state transition was not included in the model; the extracted machines did not model the output of the RNN. The algorithm quantized the state space using an SOM (as Tiňo & Šajda, 1995, also did). The generation of states and state transitions was divided into two phases: the pretest phase, where the RNN was domain driven, and a self-driven phase, where the output of the RNN was used as input in the next time-step (this RNN was trained to predict a long sequence of symbols). In Tiňo and Köteles (1999) (further described in Tiňo, Dorffner, & Schittenkopf, 2000), the SOM was replaced with a dynamic cell structure (DCS; Bruske & Sommer, 1995), but otherwise the algorithm was the same (see the summary in Table 9).

The stochastic machines are possible to analyze in new and interesting ways. The authors (Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999), for example, used entropy spectra (Young & Crutchfield, 1993) to compare the probabilities of strings generated by the RNNs with the probabilities of the strings in the original source. The results were interesting, but there were no indications in Tiňo and Köteles (1999) and Tiňo and Vojtek (1998) of how well the extracted machines corresponded to the network (i.e., rule fidelity) or how well they generalized on any test set (i.e., rule accuracy).[8] The

[8] Unless the entropy spectra analysis is considered a form of accuracy measurement.

Table 10: Neural Prediction Machines (NPMs) Differ from the FSM Ordinarily Extracted from RNNs in That State Transitions Are Not Incorporated into the Model.

Neural Prediction Machine, Vector Quantizer, Sampling on Domain (Tiňo et al., 2004)	
Rule type	NPM predicting the next output based on the current state of the RNN; state transitions not modeled
Quantization	k-means
State generation	Sampling
Networks	First-order RNN
Domains	Continuous chaotic laser data domain transformed to four symbols and recursive natural language domains

comprehensibility of the extracted rules cannot be determined from these articles. The fact that the extracted machines did not model the output of the RNN also makes it difficult to evaluate this algorithm in the context of the other RNN-RE algorithms.

Another related approach, which perhaps is not rule extraction per se but can perhaps at least be termed a partial rule extraction algorithm, is the neural prediction machine (NPM) constructed in Tiňo, Čerňanský, and Beňušková (2004). The NPM predicts the next symbol given the state of the network; the state dynamics are handled by the RNN and not extracted at all (see a summary of this approach in Table 10).

4.6 A Pedagogical Approach. All previously described algorithms fall into the category *compositional* in ADT's translucency classification (see section 3). There is to our knowledge only one algorithm that uses a pedagogical approach instead. Vahed and Omlin (Vahed & Omlin, 1999, 2004) used a machine learning method requiring only the input and the output to extract the machine; the internal state is ignored (see the summary in Table 11). The data used for extraction were based on all strings up to a given length. The input and output of the network were recorded and fed to the polynomial-time Trakhtenbrot-Barzdin algorithm (Trakhtenbrot & Barzdin, 1973).

It was also reported that this algorithm was more successful in returning correct DFAs than clustering-based algorithms (Giles, Miller, Chen, Sun, et al., 1992). This actually seems to be the only article that at all describes an experimental comparison of different RE techniques. The machine learning algorithm they used is indeed of polynomial time complexity, given that a prefix tree (see Figure 6) is available. But the size of the prefix tree up to a string length L is of complexity $O(n^L)$, where n is the number of symbols. As a consequence, this approach is likely to have some problems scaling up to more complex problems with more symbols.

Table 11: The Only RNN-RE Algorithm Where the Internal State of the RNN Is Not Regarded During the Extraction Process.

DFA Extraction, Black Box Model (Vahed & Omlin, 1999, 2004)	
Rule type	Moore DFA with binary (accept-reject) output
Quantization	N/A
State generation	All strings up to a certain length
Networks	Second-order RNN
Domains	One randomly generated 10-state DFA

4.7 RE-Supporting RNN Architectures. Clusters can be induced during training to support RE in later stages (Zeng et al., 1993; Frasconi et al., 1996). This was originally suggested in Das and Das (1991) and further developed in Das and Mozer (1994, 1998). Training to induce clusters results, if successfully performed, in RNNs that are trivially transformed to finite machines. Since the focus of this article is on the details of the extraction procedure, more details of these approaches will not be included here.

5 RNN-RE: Fool's Gold?

Kolen (1993) showed with some simple examples that some dynamic systems with real-valued state space (e.g., an RNN) cannot be described discretely without introducing peculiar results (cf. Kolen & Pollack, 1995). If you want to approximate the behavior of a physical system with a real-valued state space as a discrete machine, you will not only risk that the approximation might not be exact. A more profound effect of the approximation is that induced machines from the same physical system may belong to completely different classes of computational models, depending only on how the transformation from the real-valued space to a discrete approximation is conducted.

This critique strikes at the very heart of RNN-RE, since the quantization of the state space is a crucial element of these algorithms and RNN-RE was actually termed "fool's gold" by Kolen (1993). He pointed out that RNNs should be analyzed as dynamical systems or more specifically iterated function systems (IFSs) rather than state machines.

There are some replies to this critique, though. One simple approach is to avoid the problem by not modeling transitions at all (Tiňo et al., 2004) or even not to quantize the state space at all (Vahed & Omlin, 1999, 2004). Another response to Kolen's critique is that extraction of a state machine from an RNN has been proven to work if the underlying RNN actually robustly implements a finite state machine (Casey, 1996). However, this does not alleviate the fact that the language class for unknown RNNs cannot be

reliably recognized. But at least there is a theoretical "guarantee" that *if* there is an FSM at the bottom of an RNN, it can always be extracted in principle.

Failure of rule extraction from an RNN could therefore be an indicator that the underlying RNN is not implementing a finite state machine. One first step in this direction has been proposed by Blair and Pollack (1997). They used unbounded growth of the macrostate set under increased resolution of the equipartition quantization method as an indicator of a nonregular underlying RNN.

If we limit ourselves to real-world domains, RE will necessarily be operating on finite domains, making FSM interpretations theoretically possible at all times (although they may not be the minimal description of a domain-RNN interaction). In fact, since the focus of RNN-RE research is on FSM extraction, the question should not be whether a language class is misjudged by an RE algorithm (since extraction on the level of the class of regular languages is one of the premises), but rather how well the extracted finite machine approximates the network, as proposed by Blair and Pollack (1997). How to evaluate the fidelity of an FSM and whether this evaluation may distinguish between errors stemming from a poorly quantized state space or from a higher language class in the RNN-domain remains an open issue.

In summary, although Kolen's critique is justified, there are still reasons that further research on RE from RNNs is interesting. There is a lack of sophisticated analysis tools that can handle the complexity of RNNs, and RNN research is hampered by this fact. Although there are theoretical possibilities that RE may result in ambiguous answers about an RNN, this holds also for many other analysis techniques. For any analysis tool to be trusted, the disadvantages must be known and kept in mind when examining the results. This is precisely what makes Kolen's observations valuable for RNN-RE usage: the exposure of some of the limits of RNN-RE techniques.

6 Discussion

In this section, the described techniques will be summarized and evaluated from the perspectives of the evaluation criteria: rule type, quantization method, state generation, and network type and domain. The two criteria, portability and quality, of the ADT taxonomy (described in section 3.2) will also be discussed.

6.1 Rule Types. It is quite clear that most of the research described in section 4 has been focused on extracting traditional DFA for classification of binary strings as grammatical or ungrammatical (Giles et al., 1991; Giles, Miller, Chen, Chen, et al., 1992; Watrous & Kuhn, 1992; Zeng et al., 1993; Alquézar & Sanfeliu, 1994a; Manolios & Fanelli, 1994; Sanfeliu & Alquézar, 1995; Omlin & Giles, 1996b; Frasconi et al., 1996; Gori et al., 1998; Vahed & Omlin, 1999, 2004). Only a few DFM extraction algorithms are

used on domains with more than two output symbols (Tiňo & Šajda, 1995; Schellhammer et al., 1998). It is also interesting to notice that only three articles (Schellhammer et al., 1998; Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999) have studied DFM RNN-RE in a prediction domain, while prediction of sequences is a quite common approach in RNN research in general (e.g., Elman 1990; Alquézar & Sanfeliu, 1994b; Gers & Schmidhuber, 2001; Jacobsson & Ziemke, 2003a).

The crisp DFM do not model probabilistic properties of macrostate transitions and macrostate interpretations; that kind of information is lost in the rules, independently of whether search or sampling is used to generate states. Hence, a more expressive set of rules may be represented in stochastic FSM (Tiňo & Vojtek, 1998; Tiňo & Köteles, 1999) and the fidelity (i.e., the coherence of the rules with the RNN) of stochastic rules should in principle be higher (given the same premises, e.g., quantization) than for their deterministic counterparts. The fidelity can, however, be measured in various ways, as the term is not clearly defined, leading to possibly ambiguous results. If stochastic machines are to be used for RNN analysis, however, it is important that the extracted machines also model the output of the RNN. Such stochastic machines (Paz, 1971; Rabin, 1963) have yet to be extracted; further work is needed to realize this.

An approach to combine the best of both worlds may be to push further on the way chosen by Schellhammer et al. (1998), where probabilities were calculated and then used as heuristics for transforming the incomplete and nondeterministic machine into a deterministic and complete machine. That way, the information loss from going from the RNN to a deterministic machine could possibly be tracked.

A last, "exotic" form of rules is the NPM. The NPM is only predicting the output of the network given the state, but is not concerned with the internal mappings of states in the RNN (Tiňo et al., 2004).

6.2 State Space Quantization. Clearly, there is no consensus about how to quantize the state space. Methods that have been used are (see section 4 for more complete reference lists) regular (grid) partition (Giles et al., 1991), *k*-means (Zeng et al., 1993; Frasconi et al., 1996; Schellhammer et al., 1998; Tiňo et al. 2004, Cechin, Pechmann Simon, & Stertz, 2003), SOM (Tiňo & Šajda, 1995; Tiňo & Vojtek, 1998; Blanco et al., 2000), dynamical cell structures (Tiňo & Köteles, 1999), "other" vector quantifiers (Manolios & Fanelli, 1994), hierarchical clustering (Alquézar & Sanfeliu, 1994a), dynamically updated intervals (Watrous & Kuhn, 1992), and fuzzy clustering (Cechin et al., 2003). That makes eight different techniques, not counting small variations in implementations. Although just a fraction of existing clustering techniques have been tested (Mirkin, 1996; Jain, Murty, & Flynn 1999), it is clear that a multitude of existing clustering techniques has been used to approach the quantization problem. But what is most striking about this

multitude of techniques used is not that they are so many but that there are no studies comparing different quantization techniques to each other in the context of RNN-RE.

The two main families of clustering techniques used are vector quantization (VQ; e.g., *k*-means and SOM) and equipartition. The main difference between these, apart from VQ partitions not being of equal sizes and shapes, is that the VQ clusters are not fixed prior to the extraction but are instead adapted to fit the actually occurring state activations in the RNN. In principle, vector quantization should be able to scale up to more state nodes than the equipartition method since the number of partitions can be arbitrarily selected independent of state space dimensionality.

6.3 State Generation. There are two basic strategies for generating the states in the RNN (see section 4 for more complete reference lists): searching (Giles et al., 1991; Zeng et al., 1993; Frasconi et al., 1996) and sampling (Watrous & Kuhn, 1992; Manolios & Fanelli, 1994; Alquézar & Sanfeliu, 1994a; Tiňo and Šajda, 1995; Schellhammer et al., 1998; Tiňo & Vojtek, 1998; Tiňo et al., 2004).

As for the clustering techniques, there are no studies comparing searching- and sampling-based RNN-RE experimentally, apart from one preliminary study (Jacobsson & Ziemke, 2003b). Unlike for clustering techniques, however, it is quite easy to see at least a few of the consequences of the choice of state generation method.

First, breadth-first search will obviously have problems of scaling up to larger problems and be especially sensitive to the number of input symbols. There are also reasons to believe that for prediction networks in domains that are not completely random, many of the transitions and states generated with breadth-first search would not be relevant or ever occur in the domain (Jacobsson & Ziemke, 2003b). Machines extracted with search are, however, guaranteed to be deterministic, which may very well be desired (see the discussion in the previous section). The extraction is also guaranteed to end up with a complete machine where all possible inputs are tested on all encountered states.

RE through sampling on the domain is not guaranteed to result in deterministic and complete machines. If this is required, there is no guarantee that a certain state space quantization will result in a solution since inconsistencies might occur. A heuristic solution to this problem has been proposed in only one article (Schellhammer et al., 1998). In summary, sampling-based RE techniques may be a better strategy for extraction of stochastic rather than deterministic machines.

6.4 Network Types and Domains. The networks that have been studied using RNN-RE are in most cases relatively small networks with few state nodes. This may be due to the fact that most domains used were simple enough to allow small networks to be trained.

There are also significantly more second-order than first-order networks. Probably this is an effect of the focus on formal language domains where second-order RNNs are more commonly used than first-order networks (Goudreau, Giles, Chakradhar, & Cheng, 1994).

The investigated domains mostly require only binary string classification. More complex domains with many symbols or deep syntactical structures have not yet been tested using RNN-RE. Therefore, the applicability of these techniques is to a large degree an open issue. The importance of rule extraction as described by Andrews et al. (1995)—for example, explanation capability and verification of ANN components—is therefore less than it would have been if the techniques had been demonstrated to work on the state-of-the-art RNNs operating on the most challenging domains.

6.5 Portability. Although most RNN-RE algorithms are compositional (i.e., under the ADT translucency criteria) and have, in principle, the same requirements on the underlying RNN, there are some implicit requirements that could be interesting to bring up, especially if the existing RNN-RE algorithms are to be applied on previously unencountered RNN architectures (or other dynamical systems) and domains. Current RE techniques are preferably used on RNNs that:

- Operate in discrete time (continuous-time RNN will not work). There is, however, no known work on continuous-time RNNs in the domain of FSM-generated languages (Forcada & Carrasco, 2001).
- Have clearly defined input, state, and output nodes (randomly structured RNNs may be problematic, e.g., echo state networks; Jaeger, 2003).
- Have a fully observable state; otherwise, unobserved state nodes or noise in the observation process would disturb the extraction process since the state space would not be reliably quantized.
- Have state nodes that can be set explicitly (for search-based techniques).
- Are deterministic; otherwise, the same problem as if the state is not fully observable would occur.
- Are fixed during RE; no training of the RNN can be allowed during the RE process.
- Operate on a preferably discrete domain (or be transformed to a discrete representation prior to RE; e.g., Giles et al. 1997) since there are no means for representing continuous input data in the current types of extracted rules.

The problem of making a list of implicit requirements is that they are implicit (i.e., not readily apparent). There may therefore be other essential

requirements that we have not managed to figure out at this stage. Also, the strengths of these requirements are not clear either; some of them may be quite easily alleviated with some enhancements of current RNN-RE techniques.

6.6 Rule Quality. The quality of RNN-RE techniques is (or should be) evaluated at the level of the actual rules rather than at the level of the algorithms. Extracted rules depend not only on the algorithm but also on the underlying domain and network. Evaluation of the rule quality therefore requires extensive studies comparing different RE techniques under similar conditions. Unfortunately, such studies have not yet been conducted for most RNN-RE algorithms.

There are, in the existing corpus of papers on RNN-RE, some indirect results that provide some indications for some of the rule quality subcategories of accuracy, fidelity, consistency, and comprehensibility.

Some studies indicate that the extracted machines indeed have high accuracy since they may even be generalizing better than the underlying RNN (Giles, Miller, Chen, Chen, et al., 1992; Giles, Miller, Chen, Sun, et al., 1992, Giles & Omlin, 1993; Omlin & Giles, 1996b). There are, however, unfortunately no studies where the fidelity of the extracted rules has been tested separately from the accuracy. The studies tend to focus on networks that are quite successful on their domain, and under such circumstances, the difference between fidelity and accuracy is very small. For networks performing badly on their domain, high fidelity would, however, imply low accuracy since the errors of the network then would be replicated by the machine.

Rule consistency has not been studied extensively, although some articles touch the subject. Rules extracted from a network during training were found to fall under a sequence of equivalence classes during training (Giles, Miller, Chen, Sun, et al., 1992). This can be seen as an example of consistency since the extracted rules after a certain time of training eventually stabilized in the same equivalence class; the set of quite similar networks at the later part of the training resulted in equivalent rules. The consistency over different parameter settings (of the quantization parameter q in the equipartitioned RNN-RE algorithm) has also been proposed as an indicator of regularity in the underlying network (Blair & Pollack, 1997). These results on consistency are, however, more or less indirect.

Rule comprehensibility is an important issue to ensure further progress for RNN-RE research. After all, if the goal of extracting rules is to understand the underlying incomprehensible network, the rules should preferably be comprehensible themselves. The comprehensibility of extracted rules has not been directly evaluated. It is, however, clear that in some cases, the RNN-RE analysis has been informative in qualitative ways.

7 Open Issues

7.1 Goals with RNN-RE Development. It is clear that the development of the RNN-RE algorithms has been driven by different goals in the different articles. But what are the possible goals of RNN-RE? Possible answers could be (taken partly from Andrews et al., 1995, in the context of RE in general):

- To acquire a generic model of the domain, that is, the RNN is used merely as a tool in the acquisition process (data mining).
- To provide an explanation of the RNN.
- To allow verification or validation of the RNN with respect to some requirements (cf. software testing) and thus make new, potentially safety-critical domains possible for RNNs.
- To improve on current RNN architectures by pinpointing errors.

The appropriate measure to evaluate the success (or rule quality) of a specific instance of an RNN-RE algorithm being applied on an RNN (and domain) depends highly on which of these (or other) goals are desired (see Figure 7). For the first goal, for example, the maximization of accuracy is the prime target. In many of the articles, it is clear that accuracy is the most important aspect of rule quality. Accuracy is a good means for evaluating rule quality as long as the goal for rule extraction is to find rules that are as good as possible on the domain. For the other goals, the maximization of fidelity is likely to be more important. After all, if the network is tested with an accuracy-maximizing RE method, the result may be rules with a performance better than the network (a result confirmed by many studies). Therefore, for purposes of RNN analysis, fidelity should be the preferred quality evaluation criterion. In some cases, however, comprehensibility may be more crucial than fidelity and accuracy. In other cases, it is imaginable that the efficiency of the algorithm (in terms of execution time or required memory storage) is the primary objective. There may also be more domain-specific measures to evaluate the degree of goal achievement. The details of the resulting RNN-RE algorithm may depend on which of these goal is aimed at. Preferably, however, the one and same algorithm should be generic enough to let the user choose among the goals.

7.2 New Challenges. What should we expect from future RNN-RE algorithms? There are some challenging applications and requirements on RNN-RE algorithms that are worth suggesting.

7.2.1 Tailor-Made Quantization algorithms. The quantization algorithm is perhaps the most critical part of extracting rules from RNNs. But what characterizes a good quantization in the context of RNN-RE? It is not necessarily spatial requirements (Sharkey & Jackson, 1995), as is usually the

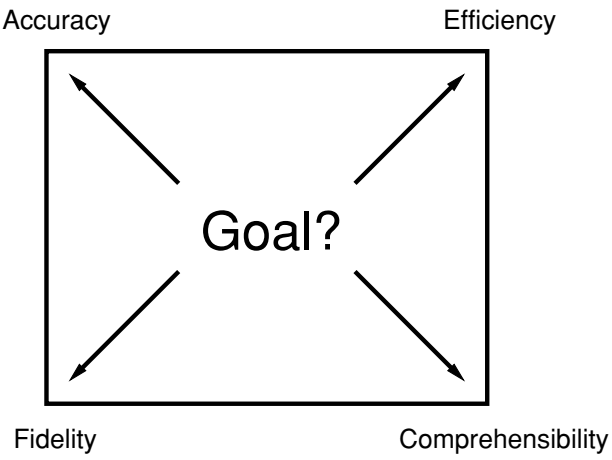

Figure 7: Four, possibly opposite, goals of RNN-RE that in an ideal algorithm would be simply be chosen by the setting of a few user-defined parameters.

case for evaluation of clustering techniques (Jain et al., 1999), but rather requirements based on properties of the extracted rule set. To have clusters that are spatially coherent and well separated is of less importance than the fidelity of the resulting rules. One would like a clustering technique where the clusters are functionally coherent and well separated rather than spatially.

If an evaluation method for quantization techniques could be defined as functional rather than spatial requirements, it could help the research on RNN-RE to find better techniques with clustering techniques tailor-made for the purposes of extracting good rules (where the definition of "good," of course, depends on the goal of the RE).

7.2.2 Goal-Oriented Gradually Refining Rule Extraction. Methods for controlling the comprehensibility-fidelity trade-off are identified as an important line of research by Craven and Shavlik (1999). This trade-off issue may be expanded to techniques where the user may, through the setting of a few parameters, not only have the ability to choose between fidelity and comprehensibility, but also fidelity and accuracy, fidelity and computation time, and others. In an ideal RNN-RE algorithm, the relation between execution time, fidelity, and comprehensibility may be as illustrated in Figure 8. Rules should be refined gradually over time, and the more time available, the higher the possibility is of acquiring rules of high fidelity and/or comprehensibility ("anytime rule extraction" (Craven & Shavlik, 1999)).

One method for gradually refining rules may be to do reextraction of uncertain or infrequent but possibly important rules by querying the network

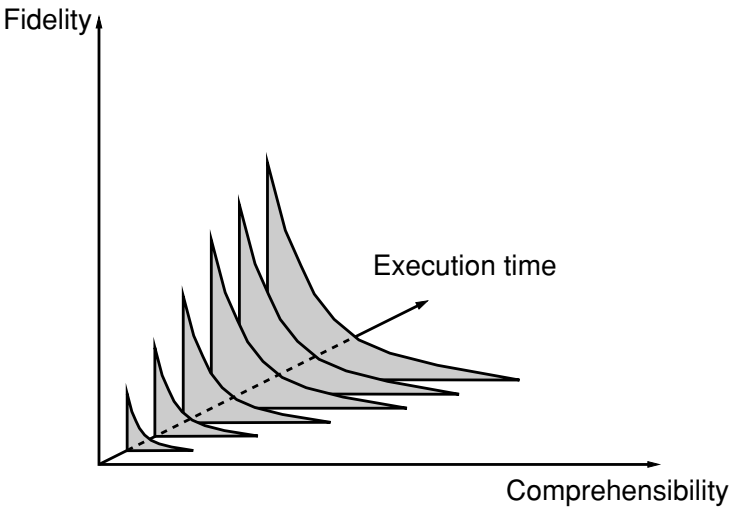

Figure 8: Relation between execution time, fidelity, and comprehensibility for ideal RNN-RE algorithms with possible gradual refinement of the rules. The more time available, the more the degree of freedom in choosing between high fidelity and comprehensibility.

(Craven & Shavlik, 1994). This can, for example, be done by directly setting states in the network to be in the vicinity of the model vector (or something equivalent) of the macrostate of interest and then testing the effect of feeding the RNN various possible inputs.

7.2.3 RNN Comparisons and Evaluations. A distance metric between RNNs could be defined by comparing rules extracted by RNN-RE. RNNs are otherwise difficult to compare directly since completely different weights can give equivalent behavior, and small differences in weights may result in very different behaviors. This sort of distance metric would be favorable when using RNN ensembles (Sharkey, 1996) to ensure a heterogeneous set of RNNs in the ensemble.

The idea that RNN-RE can be used as an indication of the complexity of the underlying RNN (or some other dynamical system) and domain could be further developed. Previous studies seem to show promising results (Crutchfield & Young, 1990; Blair & Pollack, 1997; Bakker & de Jong, 2000) with regard to complexity estimations that go beyond Shannon entropy and Kolmogorov complexity. A simpler version of the complexity issue would be to ask to which degree the recurrent part of the network affects the behavior of the network (i.e., whether the RNN is equivalent to any nonrecurrent network).

7.2.4 RNN Debugging. The underlying task of the RNN (e.g., prediction) can be integrated with the rules in order to identify more exactly when and how erroneous behavior occurs in the network. This can be done simply by marking which states in the extracted machines are involved in the errors. This error can perhaps then be further traced back, in the rules, to the actual erroneous behavior. This can perhaps be further integrated with the training of the network by letting the trainer update the weights only in situations identified by the rules as being part of an erroneous behaviour (cf. Schmidhuber, 1992).

7.3 Some Practical Hints. Since this article is, in part, aimed at attracting more researchers to the field, it is perhaps a good idea to not just identify open issues, but also to give some practical hints about how things should be done. These hints are basically a repetition of Craven and Shavlik (1999), but they are important enough to be repeated.

First, if you are developing a new RNN-RE algorithm, strive for generality (i.e., high portability). The usefulness of the algorithm you develop will directly correlate with how easily it can be used on RNNs already developed, implemented, and tested originally without intentions of making them suitable for RE. In fact, Craven and Shavlik (1999) even suggested that the RE algorithms should be made so general that not even the assumption that the underlying system is a neural network at all is necessary. In fact, some things indicate that this is already a fact for most RNN-RE algorithms, considering the very limited assumptions of the underlying RNN (cf. section 2.1).

Another good piece of advice is to seek out collaborators who already have RNNs they want to analyze (Craven & Shavlik, 1999). It is highly unlikely that you will have time to develop both state-of-the-art RNNs and state-of-the-art RNN-RE algorithms at the same time. Finding willing collaborators should not be too difficult since researchers applying novel RNNs on new domains will most likely benefit from the knowledge acquired through rule extraction.

Another important ingredient for making a technique attractive is to make implementations of the techniques publicly available (Craven & Shavlik, 1999). After all, the techniques are aimed at being used by researchers that are quite busy pursuing their own line of work.

8 Conclusions

Ideally, if the RNN-RE techniques developed so far had been really successful, they would be among the first analysis techniques to be used when new RNN architectures had been developed or a new domain had been concurred. No other analysis tools seem to promise a more detailed and profound understanding of RNNs. But we are not there yet.

Despite numerous achievements, there seems to be no apparent common direction (or well-defined goals) in the RNN-RE research community. In most cases, developed algorithms do not seem to be built based on previous results, and there seems to be a very slow (if any) progress toward handling more complex RNNs and domains with RNN-RE algorithms. In fact, only one algorithm has been used to any wide extent in the follow-up work, and, moreover, it is the first RNN-RE algorithm developed (Giles, Miller, Chen, Chen, et al., 1992). It is surprising that it has not been replaced by anything significantly better. Actually, later algorithms may very well be better, but they are still not used as frequently as the first one, and there are almost no comparative studies.

This article identifies important research issues that, if addressed, would help give this field a push forward. More important, goals against which any progress in this field should be measured have been suggested. Further clarity about what these goals constitute is needed, and they need to be constantly updated, but the main issue is that the lack of common goals is alleviated bit by bit. A natural next step would be to find standardized problems on which rule extraction techniques can be compared. Preferably, the standardized problems would not be bound to only RNN problems but also include other dynamical systems problems.

RNNs, and neural networks in general, are, as they are simulated entities, very "studyable" once we have tools to study them. And the algorithms reviewed here may hold the seed for a deeper and more general notion of analysis than seen before. Better analysis tools may in turn help RNN research to progress more rapidly once we get a deeper understanding of what the networks are actually doing. In many other disciplines of science, the quantum leaps in progress often stem from more sophisticated analysis tools and measuring devices producing qualitatively new data conflicting with existing models (anomalies) that eventually may result in scientific revolutions (Kuhn, 1962). Today we have deep but partially conflicting theories of what the RNNs will be able to do in practice (i.e., the Turing machine equivalence versus the difficulty of acquiring correct behavior through learning), but we have no means for evaluating what particular instances of RNNs are actually doing in an efficient manner.

With critical eyes, rule extraction from recurrent neural networks may seem an infinitesimal subfield within another infinitesimal subfield and thereby with a very limited potential to deliver interesting scientific results. But if there is a future microscope for zooming in on RNNs, I would hold that there are good reasons to believe that rule extraction mechanisms will be the operational parts, or lenses, of that microscope. And like a real-world microscope, this RNN microscope will, if general enough, be able to zoom in on other types of dynamical systems and physical computing devices and thus contribute to the scientific community in a considerably broader sense.

Acknowledgments

First, I thank my librarian, Karin Lundberg, for her efficient and tireless efforts of providing me with all papers. Without her, this survey would not have been possible. I also thank André Grüning, Amanda Sharkey, Ron Sun, and, especially, Tom Ziemke for commenting on early versions of this article and also Gunnar Buason, Anders Jacobsson, and Claudina Riguetti for proofreading a later version. I also thank several of the cited authors for helping me make the list of references more complete and for engaging in some very interesting email discussions. Thanks also to the two anonymous reviewers whose suggestions helped me to streamline the article significantly.

References

Alquézar, R., & Sanfeliu, A. (1994a). A hybrid connectionist symbolic approach to regular grammar inference based on neural learning and hierarchical clustering. In *Proceedings of ICGI'94* (pp. 203–211). Berlin: Springer-Verlag.

Alquézar, R., & Sanfeliu, A. (1994b). Inference and recognition of regular grammars by training recurrent neural networks to learn the next-symbol prediction task. In F. Casacuberta & A. Sanfeliu (Eds.), *Advances in pattern recognition and applications*: *Selected papers from the Vth Spanish Symposium on Pattern Recognition and Image Analysis* (pp. 48–59). Singapore: World Scientific.

Alquézar, R., Sanfeliu, A., & Sainz, M. (1997). Experimental assessment of connectionist regular inference from positive and negative examples. In *VII Simposium Nacional de Reconocimiento de Formas y Análisis de Imágenes* (Vol. 1, pp. 49–54). Barcelona, Spain.

Andrews, R., Diederich, J., & Tickle, A. (1995). Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, *8*(6), 373–389.

Bakker, B. (2004). *The state of mind: Reinforcement learning with recurrent neural networks*. Unpublished doctoral dissertation, Leiden University.

Bakker, B., & de Jong, M. (2000). The epsilon state count. In J. A. Meyer, A. Berthoz, D. Floreano, H. Roitblat, & S. Wilson (Eds.), *From animals to animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior* (pp. 51–60). Cambridge, MA: MIT Press.

Barreto, G. A., Araújo, A. F. R., & Kremer, S. C. (2003). A taxonomy for spatiotemporal connectionist networks revisited: The unsupervised case. *Neural Computation*, *15*(6), 1255–1320.

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

Blair, A., & Pollack, J. (1997). Analysis of dynamical recognizers. *Neural Computation*, *9*(5), 1127–1142.

Blanco, A., Delgado, M., & Pegalajar, M. C. (2000). Extracting rules from a (fuzzy/crisp) recurrent neural network using a self-organizing map. *International Journal of Intelligent Systems*, *15*, 595–621.

Bodén, M., Jacobsson, H., & Ziemke, T. (2000). Evolving context-free language predictors. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1033–1040). San Mateo, CA: Morgan Kaufmann.

Bodén, M., Wiles, J., Tonkes, B., & Blair, A. (1999). Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units. In *Proceedings of ICANN 99* (pp. 359–364). Piscataway, NJ: IEEE.

Bruske, J., & Sommer, G. (1995). Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, *7*, 845–865.

Bullinaria, J. A. (1997). Analyzing the internal representations of trained artificial neural networks. In A. Browne (Ed.), *Neural network analysis, architectures and applications*. London: IOP Publishing, (pp. 3–26).

Carrasco, R. C., & Forcada, M. L. (2001). Simple strategies to encode tree automata in sigmoid recursive neural networks. *IEEE Transactions on Knowledge and Data Engineering*, *13*(2), 148–156.

Carrasco, R. C., Forcada, M. L., Muñoz, M. A. V., & Neco, R. P. (2000). Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units. *Neural Computation*, *12*(9), 2129–2174.

Casey, M. (1996). The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, *8*(6), 1135–1178.

Cechin, A. L., Pechmann Simon, D. R., & Stertz, K. (2003). State automata extraction from recurrent neural nets using k-means and fuzzy clustering. In *XXIII International Conference of the Chilean Computer Science Society* (pp. 73–78). Piscataway, NJ: IEEE Computer Society.

Cicchello, O., & Kremer, S. C. (2003). Inducing grammars from sparse data sets: A survey of algorithms and results. *Journal of Machine Learning Research*, *4*, 603–632.

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

Craven, M. C., & Shavlik, J. W. (1994). Using sampling and queries to extract rules from trained neural networks. In W. W. Cohen & H. Hirsh (Eds.), *Machine learning: Proceedings of the Eleventh International Conference*. San Francisco: Morgan Kaufmann.

Craven, M. W., & Shavlik, J. W. (1996). Extracting tree-structured representations of trained networks. In D. Touretzky, M. Mozer, & M. Hasselmo (Eds.), *Advances in neural information processing systems*, *8* (pp. 24–30). Cambridge, MA: MIT Press.

Craven, M. W., & Shavlik, J. W. (1999). *Rule extraction: Where do we go from here*? (Tech. Rep. Machine Learning Research Group Working Paper 99-1). Madison: Department of Computer Sciences, University of Wisconsin.

Crutchfield, J. P. (1994). The calculi of emergence: Computation, dynamics, and induction. *Physica D*, *75*, 11–54.

Crutchfield, J., & Young, K. (1990). Computation at the onset of chaos. In W. Zurek (Ed.), *Complexity, entropy and the physics of information*. Reading, MA: Addison-Wesley.

Das, S., & Das, R. (1991). Induction of discrete-state machine by stabilizing a simple recurrent network using clustering. *Computer Science and Informatics*, *21*(2), 35–40.

Das, S., Giles, C. L., & Sun, G. Z. (1993). Using prior knowledge in a NNPDA to learn context-free languages. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.),

Advances in neural information processing systems, *5* (pp. 65–72). San Mateo, CA: Morgan Kaufmann.

Das, S., & Mozer, M. C. (1994), A unified gradient-descent/clustering architecture for finite state machine induction. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems*, *6* (pp. 19–26). San Mateo, CA: Morgan Kaufmann.

Das, S., & Mozer, M. (1998). Dynamic on-line clustering and state extraction: An approach to symbolic learning. *Neural Networks*, *11*(1), 53–64.

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

Fanelli, R. (1993). *Grammatical inference and approximation of finite automata by Elman type recurrent neural networks trained with full forward error propagation* (Tech. Rep. No: NNRG930628A). Brooklyn: Department of Physics, Brooklyn College of the City University of New York.

Forcada, M. L. (2002). *Neural networks: Automata and formal models of computation. An unfinished survey*. Available online at: http://www.dlsi.ua.es/∼mlf/nnafmc/.

Forcada, M. L., & Carrasco, R. C. (2001). Finite-state computation in analog neural networks: Steps towards biologically plausible models? In S. Wermter, J. Austin, & D. Willshaw (Eds.), *Emergent computational models based on neuroscience*. Berlin: Springer-Verlag.

Frasconi, P., Gori, M., Maggini, M., & Soda, G. (1996). Representation of finite state automata in recurrent radial basis function networks. *Machine Learning*, *23*(1), 5–32.

Gers, F. A., & Schmidhuber, J. (2001). LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks*, *12*(6), 1333–1340.

Giles, C. L., Chen, D., Miller, C., Chen, H., Sun, G., & Lee, Y. (1991). Second-order recurrent neural networks for grammatical inference. In *Proceedings of International Joint Conference on Neural Networks* (Vol. 2, pp. 273–281). Piscataway, NJ: IEEE.

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

Giles, C. L., Lawrence, S., & Tsoi, A. (1997). Rule inference for financial prediction using recurrent neural networks. In *Proceedings of IEEE/IAFE Conference on Computational Intelligence for Financial Engineering (CIFEr)* (pp. 253–259). Piscataway, NJ: IEEE.

Giles, C. L., Lawrence, S., & Tsoi, A. C. (2001). Noisy time series prediction using a recurrent neural network and grammatical inference. *Machine Learning*, *44*(1/2), 161–183.

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

Giles, C. L., Miller, C. B., Chen, D., Sun, G. Z., Chen, H. H., & Lee, Y. C. (1992). Extracting and learning an unknown grammar with recurrent neural networks. In J. E. Moody, S. J. Hanson, & R. P. Lippmann (Eds.), *Advances in neural information processing systems*, *4* (pp. 317–324). San Francisco: Morgan Kaufmann.

Giles, C. L., & Omlin, C. W. (1993). Extraction, insertion and refinement of symbolic rules in dynamically driven recurrent neural networks. *Connection Science*, *5*(3–4), 307–337.

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

Golea, M. (1996). *On the complexity of rule extraction from neural networks and network-querying* (Tech. Rep.). Canberra: Australian National University.

Gori, M., Maggini, M., Martinelli, E., & Soda, G. (1998). Inductive inference from noisy examples using the hybrid finite state filter. *IEEE Transactions on Neural Networks*, *9*(3), 571–575.

Gori, M., Maggini, M., & Soda, G. (1994). Scheduling of modular architectures for inductive inference of regular grammars. In *'ECAI'94 Workshop on Combining Symbolic and Connectionist Processing, Amsterdam* (pp. 78–87). New York: Wiley.

Goudreau, M. W., & Giles, C. L. (1995). Using recurrent neural networks to learn the structure of interconnection networks. *Neural Networks*, *8*(5), 793–804.

Goudreau, M. W., Giles, C. L., Chakradhar, S. T., & Chen, D. (1994). First-order vs. second-order single layer recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*(3), 511–518.

Hammer, B., & Tiňo, P. (2003). Recurrent neural networks with small weights implement definite memory machines. *Neural Computation*, *15*(8), 1897–1929.

Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, *46*(1–2), 47–75.

Hopcroft, J., & Ullman, J. D. (1979). *Introduction to automata theory, languages, and compilation*. Reading, MA: Addison-Wesley.

Horne, B. G., & Giles, C. L. (1995). An experimental comparison of recurrent neural networks. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems, 7'* (pp. 697–704). Cambridge, MA: MIT Press.

Horne, B. G., & Hush, D. R. (1994). Bounds on the complexity of recurrent neural network implementations of finite state machines. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems*, *6* (pp. 359–366). San Mateo, CA: Morgan Kaufmann.

Husbands, P., Harvey, I., & Cliff, D. T. (1995). Circle in the round: State space attractors for evolved sighted robots. *Robotics and Autonomous Systems*, *15*(1–2), 83–106.

Jacobsson, H., & Ziemke, T. (2003a). Improving procedures for evaluation of connectionist context-free language predictors. *IEEE Transactions on Neural Networks*, *14*(4), 963–966.

Jacobsson, H., & Ziemke, T. (2003b). *Reducing complexity of rule extraction from prediction RNNs through domain interaction* (Tech. Rep. No. HS-IDA-TR-03-007). Skövde: Department of Computer Science, University of Skövde, Sweden.

Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Advances in neural information processing systems, 15* (pp. 593–600). Cambridge, MA: MIT Press.

Jagota, A., Plate, T., Shastri, L., & Sun, R. (1999). Connectionist symbol processing: Dead or alive? *Neural Computing Surveys*, *2*, 1–40.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys*, *31*(3), 264–323.

Kohonen, T. (1995). *Self-organizing maps*. Berlin: Springer.

Kolen, J. F. (1993). Fool's gold: Extracting finite state machines from recurrent network dynamics. In J. Cowan, G. Tesauro, & J. Alspector (Eds.), *Neural information processing systems, 6* (pp. 501–508). San Mateo, CA: Morgan Kaufmann.

Kolen, J. F. (1994). *Exploring the computational capabilities of recurrent neural networks*. Unpublished doctoral dissertation, Ohio State University.

Kolen, J. F., & Kremer, S. C. (Eds.). (2001). *A field guide to dynamical recurrent networks*. Piscataway, NJ: IEEE Press.

Kolen, J., & Pollack, J. (1995). The observers' paradox: Apparent computational complexity in physical systems. *Journal of Exp. and Theoret. Artificial Intelligence*, *7*(3), 253–277.

Kremer, S. C. (2001). Spatiotemporal connectionist networks: A taxonomy and review. *Neural Computation*, *13*(2), 248–306.

Kuhn, T. S. (1962). *The structure of scientific revolutions*. Chicago: University of Chicago Press.

Lawrence, S., Giles, C. L., & Fong, S. (2000). Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, *12*(1), 126–140.

Lawrence, S., Giles, C. L., & Tsoi, A. C. (1998). Symbolic conversion, grammatical inference and rule extraction for foreign exchange rate prediction. In Y. Abu-Mostafa, A. S. Weigend, and P. Refenes (Eds.), *Neural networks in the capital markets NNCM96* (pp. 333–345). Singapore: World Scientific.

Maggini, M. (1998). Recursive neural networks and automata. In C. L. Giles & M. Gori (Eds.), *Adaptive processing of sequences and data structures* (pp. 248–295). Berlin: Springer-Verlag.

Manolios, P., & Fanelli, R. (1994). First order recurrent neural networks and deterministic finite state automata. *Neural Computation*, *6*(6), 1155–1173.

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

Medler, D. (1998). A brief history of connectionism. *Neural Computing Surveys*, *1*(1), 61–101.

Meeden, L. A. (1996). An incremental approach to developing intelligent neural network controllers for robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *26*(3), 474–485.

Miller, C. B., & 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.

Mirkin, B. (1996). *Mathematical classification and clustering*. Norwood, MA: Kluwer.

Niklasson, L., & Bodén, M. (1997). Representing structure and structured representations in connectionist networks. In A. Browne (Ed.), *Neural Network perspectives on cognition and adaptive robotics* (pp. 20–50). London: IOP Press.

Omlin, C. W. (2001). Understanding and explaining DRN behaviour. In J. F. Kolen & S. C. Kremer (Eds.), *A field guide to dynamical recurrent networks* (pp. 207–228). Piscataway, NJ: IEEE Press.

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, CA: Morgan Kaufmann.

Omlin, C. W., & Giles, C. L. (1996a). Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, *43*, 937–972.

Omlin, C. W., & Giles, C. L. (1996b). Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, *9*(1), 41–51.

Omlin, C. W., & Giles, C. L. (1996c). Rule revision with recurrent neural networks. *Knowledge and Data Engineering*, *8*(1), 183–188.

Omlin, C. W., & Giles, C. L. (2000). Symbolic knowledge representation in recurrent neural networks: Insights from theoretical models of computation. In I. Cloete & J. M. Zurada (Eds.), *Knowledge-based neurocomputing*. Cambridge, MA: MIT Press.

Omlin, C. W., Giles, C., & Miller, C. (1992). Heuristics for the extraction of rules from discrete-time recurrent neural networks. In *Proceedings of the International Joint Conference on Neural Networks* (Vol. 1, pp. 33–38). New York: International Neural Network Society, IEEE.

Omlin, C. W., Thornber, K. K., & Giles, C. L. (1998). Deterministic fuzzy finite state automata can be deterministically encoded into recurrent neural networks. *IEEE Transactions on Fuzzy Systems*, *6*(1), 76–89.

Paz, A. (1971). *Introduction to probabilistic automata*. Orlando, FL: Academic Press.

Pollack, J. B. (1987). Cascaded back-propagation on dynamic connectionist networks. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society* (pp. 391–404). Mahwah, NJ: Erlbaum.

Rabin, M. O. (1963). Probabilistic automata. *Information and Control*, *6*, 230–245.

Rodriguez, P. F. (1999). *Mathematical foundations of simple recurrent neural networks in language processing*. Unpublished doctoral dissertation, University of California, San Diego.

Rodriguez, P., Wiles, J., & Elman, J. L. (1999). A recurrent network that learns to count. *Connection Science*, *11*, 5–40.

Sanfeliu, A., & Alquézar, R. (1995). Active grammatical inference: A new learning methodology. In *Shape, Structure and Pattern Recognition, 5th IAPR International Workshop on Structural and Syntactic Pattern Recognition* (pp. 191–200). Singapore: World Scientific.

Schellhammer, I., Diederich, J., Towsey, M., & Brugman, C. (1998). Knowledge extraction and recurrent neural networks: An analysis of an Elman network trained on a natural language learning task. In D. M. W. Powers (Ed.), *Proceedings of the Joint Conference on New Methods in Language Processing and Computational Natural Language Learning: NeMLaP3/CoNLL98'* (pp. 73–78). Somerset, NJ: Association for Computational Linguistics.

Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, *4*(2), 234–242.

Servan-Schreiber, D., Cleeremans, A., & McClelland, J. L. (1989). Learning sequential structure in simple recurrent networks. In D. S. Touretzky (Ed.), *Advances in neural information processing systems*, *1* (pp. 643–652). San Mateo, CA: Morgan Kaufmann.

Servan-Schreiber, D., Cleeremans, A., & McClelland, J. L. (1991). Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, *7*, 161–193.

Sharkey, A. J. C. (Ed.). (1996). [Special issue]. Combining artificial neural nets: Ensemble approaches *Connection Science*, *8* (3–4).

Sharkey, N. E., & Jackson, S. A. (1995). An internal report for connectionists. In R. Sun & L. A. Bookman (Eds.), *Computational architectures integrating neural and symbolic processes* (pp. 223–244). Norwood, MA: Kluwer.

Siegelmann, H. T., & Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computer and System Sciences*, *50*(1), 132–150.

Sima, J., & Orponen, P. (2003). General purpose computation with neural networks: A survey of complexity theoretic results. *Neural Computation*, *15*, 2727–2778.

Sun, G. Z., Giles, C. L., & Chen, H. H. (1998). The neural network pushdown automation: Architecture, dynamics and learning. In C. Giles & M. Gori (Eds.), *Adaptive processing of sequences and data structures* (pp. 296–345). Berlin: Springer.

Sun, R., & Giles, C. L. (Eds.), (2001). *Sequence learning: Paradigms, algorithms, and applications*. Berlin: Springer.

Sun, R., Peterson, T., & Sessions, C. (2001). The extraction of planning knowledge from reinforcement learning neural networks. In *Proceedings of WIRN'2001*. Berlin: Springer-Verlag.

Tabor, W., & Tanenhaus, M. (1999). Dynamical models of sentence processing. *Cognitive Science*, *24*(4), 491–515.

Tickle, A., Andrews, R., Golea, M., & Diederich, J. (1997). Rule extraction from artificial neural networks. In A. Browne (Ed.), *Neural network analysis, architectures and applications* (pp. 61–99). London: IOP Publishing.

Tickle, A. B., Andrews, R., Golea, M., & Diederich, J. (1998). The truth will come to light: Directions and challenges in extracting the knowledge embedded within mined artificial neural networks. *IEEE Transactions on Neural Networks*, *9*(6), 1057–1068.

Tiňo, P., Čerňanský, M., & Beňušková, L. (2004). Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, *15*(1), 6–15.

Tiňo, P., Dorffner, G., & Schittenkopf, C. (2000). Understanding state space organization in recurrent neural networks with iterative function systems dynamics. In S. Wermter & R. Sun (Eds.), *Hybrid neural symbolic integration* (pp. 256–270). Berlin: Springer-Verlag.

Tiňo, P., & Hammer, B. (2003). Architectural bias in recurrent neural networks—Fractal analysis. *Neural Computation*, *15*(8), 1931–1957.

Tiňo, P., Horne, B. G., Giles, C. L., & Collingwood, P. C. (1998). Finite state machines and recurrent neural networks—automata and dynamical systems approaches. In J. E. Dayhoff & O. Omidvar (Eds.), *Neural networks and pattern recognition* (pp. 171–220). Orlando, FL: Academic Press.

Tiňo, P., & Köteles, M. (1999). Extracting finite-state representations from recurrent neural networks trained on chaotic symbolic sequences. *IEEE Transactions on Neural Networks*, *10*(2), 284–302.

Tiňo, P., & Šajda, J. (1995). Learning and extracting initial mealy automata with a modular neural network model. *Neural Computation*, *7*(4), 822–844.

Tiňo, P., & Vojtek, V. (1998). Extracting stochastic machines from recurrent neural networks trained on complex symbolic sequences. *Neural Network World*, *8*(5), 517–530.

Tomita, M. (1982). Dynamic construction of finite-state automata from examples using hillclimbing. In *Proceedings of Fourth Annual Cognitive Science Conference* (pp. 105–108).

Tonkes, B., Blair, A., & Wiles, J. (1998). Inductive bias in context-free language learning. In T. Downs, M. Frean, & M. Gallagher (Eds.), *Proceedings of the Ninth Australian Conference on Neural Networks*. Brisbane: Department of Computer Science and Electrical Engineering, University of Queensland.

Tonkes, B., & Wiles, J. (1999). Learning a context-free task with a recurrent neural network: An analysis of stability. In R. Heath, B. Hayes, A. Heathcote, & C. Hooker (Eds.), *Dynamical Cognitive Science: Proceedings of the Fourth Biennial Conference of the Australasian Cognitive Science Society*. University of Newcastle.

Towell, G. G., & Shavlik, J. W. (1993). The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, *13*(1), 17–101.

Trakhtenbrot, B. A., & Barzdin, J. M. (1973). *Finite automata: Behavior and synthesis*. Amsterdam: North-Holland.

Vahed, A., & Omlin, C. W. (1999). Rule extraction from recurrent neural networks using a symbolic machine learning algorithm (Tech. Rep. No. US-CS-TR-4). Stellenbosch, South Africa, University of Stellenbosch.

Vahed, A., & Omlin, C. W. (2004). A machine learning method for extracting symbolic knowledge from recurrent neural networks. *Neural Computation*, *16*, 59–71.

Watrous, R. L., & Kuhn, G. M. (1992). Induction of finite-state automata using second-order recurrent networks. In J. E. Moody, S. J. Hanson, & R. P. Lippmann (Eds.), *Advances in neural information processing systems*, *4* (pp. 309–317). San Mateo, CA: Morgan Kaufmann.

Wiles, J., & Elman, J. L. (1995). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent neural networks. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society* (pp. 482–487). Cambridge MA: MIT Press.

Young, K., & Crutchfield, J. P. (1993). Fluctuation spectroscopy. *Chaos, Solutions, and Fractals*, *4*, 5–39.

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

Ziemke, T., & Thieme, M. (2002). Neuromodulation of reactive sensorimotor mappings as a short-term memory mechanism in delayed response tasks. *Adaptive Behavior*, *10*(3/4), 185–199.

Received April 23, 2004; accepted November 2, 2004.

Copyright of Neural Computation is the property of MIT Press and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.