Reducing Complexity of Rule Extraction from Prediction RNNs trough Domain Interaction

Henrik Jacobsson
Department of Computer Science
University of Skövde, Sweden
henrik.jacobsson@ida.his.se

Tom Ziemke
Department of Computer Science
University of Skövde, Sweden
tom.ziemke@ida.his.se

Abstract

This paper presents a quantitative investigation of the differences between rule extraction through breadth first search and through sampling the states of the RNN in interaction with its domain. We show that for an RNN trained to predict symbol sequences in formal grammar domains, the breadth first search is especially inefficient for languages sharing properties with realistic real world domains. We also identify some important research issues, needed to be resolved to ensure further development in the field of rule extraction from RNNs.

1 Introduction

An RNN can be painstakingly difficult to analyze. Very often RNN analysis becomes a matter of creating small enough networks to allow a direct visualization of the internal activations. There are almost as many approaches to RNN analysis as there are papers about RNN and the methods are often *ad hoc* and adapted to specific domains and network architectures. Rule extraction (RE) from RNNs [1, 2, 3, 4, 5] offers a very promising tool for analyzing RNNs as it generates a functional model (usually a finite state automaton, FSA) of the of the RNN, providing an abstract symbolic model of the potentially complex analog network dynamics. In comparison to other analysis tools, such as cluster analysis, vector flow fields, analysis of fixed points etc., RE gives insight not only to the "passive" clusters resulting in the state space, but also to the "active" role of these clusters in the RNN interaction with the domain. RE is also not inherently limited by the dimensionality of the state space as are visualization methods. However, RE suffers from an apparent increasing space and time complexity for larger and more complex networks and therefore various heuristics need to be developed to allow RE to tackle more 'difficult' RNNs.

The effect of one such heuristic will be investigated quantitatively in this paper. The complexity of the behavior of an RNN is a product of its internal functional mappings generating sequences of states and output and of the complexity of the domain from which the network is fed input patterns. It is well known that even relatively simple systems can exhibit surprisingly complex behavior in interaction with a complex environment but the opposite is true also: the complexity of the behavior of a potentially complex system can be restricted by a simple environment. We will in this paper show an example of how this can be exploited as a heuristics for RE from RNNs by using the domain as a means for generating the states of the network that are the basis for the extracted rule set as opposed

to performing a breadth first search based on the possible input patterns. Both methods have been used previously in RE algorithms, but to our knowledge no comparative study has been presented.

We will first introduce our definition of RNNs, rule extraction and some theoretical prerequisits. Then the experiments and results are presented. In the last section we draw some conclusions and discuss possible future directions.

2 Background

In this paper we will, for simplicity, stick to a very simple definition of recurrent neural networks. The activations of the input, state and output nodes are for example restricted to values in the interval $[0, 1]$ and the output is functionally dependent on the state alone (excluding for example some forms of second ordered networks).

Definition 1 A *Recurrent Neural Network* is a 6-tuple $R = \langle I, O, S, \delta, \gamma, \mathbf{s}_0 \rangle$ where
$I \subseteq [0, 1]^{n_I}$ is the *input space*,
$S \subseteq [0, 1]^{n_S}$ is the *state space*,
$O \subseteq [0, 1]^{n_O}$ is the *output space*,
$\delta : S \times I \to S$ is the *state transition function* and
$\gamma : S \to O$ is the *state interpretation function*
$\mathbf{s}_0 \in S$ is the initial state vector

Where n_I, n_S and n_O are the dimensionality of each respective space. Note that the weights of the connections and activation functions of the individual nodes are subsumed by δ and γ in this definition. Those details are simply ignored by existing RE algorithms and the neurons of the network are treated as ensembles rather than as individuals. The term *compositional* was suggested by [6] to denote this level of granularity of the rule extraction algorithm's view upon the underlying network. The other modes of granularity are decompositional (white box), pedagogical (black box) and eclectic (containing elements of both decompositional and pedagogical).

States in the state space S will be visited when the network is fed input vectors from the input space. However, the full set of possible input patterns is seldomly needed to take into account for training or analysis of the RNN, e.g. if different input features are strongly correlated. Instead we can define the set $\hat{I} \subset I$ as a *finite* set of patterns that the network actually will receive *in situ*, i.e. when receiving input from the domain. We have here chosen to define $\hat{I}$ as finite since in previous approaches to RE from RNNs, formal language tasks have almost exclusively been considered. For this reason we introduce a set of symbols, Σ, isomorphic to $\hat{I}$, i.e. for every symbol in Σ, there is exactly one corresponding member in $\hat{I}$. In many papers where a formal language recognition/prediction task is studied, the symbols of Σ are encoded in I through 'one hot' encoding, i.e. every symbol of Σ 'activates' only one corresponding element of the input vector.

When the network is fed patterns from $\hat{I}$ a number of states will be visited. This set can formally be defined as the set of $\hat{I}$*-accessible states* from the initial state $\mathbf{s}_0$, let us call it $\mathcal{A}_0^{\hat{I}} \subseteq S$. $\mathcal{A}_0^{\hat{I}}$ is composed of those states in S that will be visited through the iterative mappings induced by *all possible input patterns* in $\hat{I}$ in *all possible orders* as defined in equations 1 and 2. In other words, $\mathcal{A}_0^{\hat{I}}$ is the set of states that would be visited if all possible sequences over Σ (denoted Σ^*) were fed to the network (with the network reset to s_0 before each new string).

$$\mathcal{Y}_0 = \{\mathbf{s}_0\}, \quad \mathcal{Y}_{n+1} = \mathcal{Y}_n \cup \bigcup_{\mathbf{i} \in \hat{I}} \delta(\mathbf{i}, \mathcal{Y}_n) \text{ for } n \geq 0 \tag{1}$$

$$\mathcal{A}_0^{\hat{I}} = \lim_{n\to\infty} \mathcal{Y}_n \tag{2}$$

A similar definition (for binary languages only) of accessible states is found in [4]. The production of these states is equivalent to that of an iterated function system (IFS) [7].

In rule extraction algorithms the state space needs to be quantized to a finite set of classes. This quantization function is here denoted $Q : S \to \{0, 1, 2, \ldots, N\}$ in its general form. In previous RE approaches Q is typically a simple orthogonal lattice dividing the state space into hypercubes (e.g. [1]), dividing the activation range of each individual state dimension into q intervals of equal size. This results in q^{n_S} hypercubes that can be uniquely enumerated. In this paper we refer to these hypercubes as *bins* and the degree of quantization in each dimension of the state space will be referred to as q. Other clustering methods used for RE from RNNs are for example k-means clustering (e.g. [2]) or a self organizing map SOM (e.g. [3]).

2.1 Rule extraction through breadth first search

One of the most common algorithm for rule extraction from RNNs is that of Giles et al. [1]. The algorithm conducts a breadth first search in the state space to extract a finite state machine from the RNN. The RNNs were prior to RE trained to classify strings as grammatical or non-grammatical. In the general case, any string in Σ^* should then be possible for the network to process.

The algorithm starts with an initial state s_o and generates the outgoing transitions from this state by computing all new states for all input symbols, i.e. $\delta(\mathbf{s}_0, \mathbf{i})$ for all $\mathbf{i} \in \hat{I}$. This is then repeated for all first states in each visited bin until all these states have been tested in this way and no new bin is visited. The number of the bin and the corresponding output of the first encountered state vector of each bin is then transformed to the extracted FSA. This FSA is then minimized using a standard minimization algorithm [8]. The RE algorithm starts with a small q and is repeated with increased values of q until the machine is consistent with the training data.

One way to view this algorithm is to see that the search generates a tree of symbols that generates a set of states in the network. From the root node (equivalent to the initial state of the network) all symbols expand to subtrees that are expanded likewise until all leaf-nodes lead to loops in $\hat{S}$. From the root node the path to each leaf node is the equivalent to a string of symbols. If all these substrings are fed to the network with a network reset between each string, the exact same states as visited during breadth first RE will be visited. This set of substrings will be called X_B, where $X_B \subset \Sigma^*$ and the states visited during the extraction of rules will be called $\mathcal{A}_0^{X_B}$, i.e. the set accessible from the initial state $\mathbf{s_0}$ through breadth first search RE, $\mathcal{A}_0^{X_B} \subseteq \mathcal{A}_0^{\hat{I}}$.

2.2 Rule extraction in a domain context

As mentioned above, in many tasks the full set of strings in Σ^* is not relevant for the training of the network. Much research on RNN is focused on *prediction tasks* which in many ways are much less restrictive than classification tasks since the role of an external "teacher" is reduced to a minimum. For prediction tasks the network is not required to correctly predict *all* possible sequences of symbols, but only the ones that belongs to the domain. The network does typically not even need to correctly predict all symbols of the sequences in the domain, as some subparts of the sequence may be inherently unpredictable. The temporal XOR problem is one such example where only every third symbol is at all predictable [9]. This means that the rules extracted from the network need only incorporate the sequences and subsequences that the network will encounter in the domain. If the network is for example trained to predict events that results from the behavior of a

autonomous robot it would not be reasonable to extract rules for actions that would never be carried out in certain situations, e.g. the event 'drive-forward' should not occur if the robot is in the state 'wall-ahead' and is successfully avoiding obstacles.

We will use the notation $X \in \Sigma^*$ to refer to a sequence generated or sampled from the domain. The sequence is written as $x_0 x_1 x_2 \dots x_n$. This domain specific input sequence will generate a sequence of states in the network which we will refer to as the X*-accessible set* from $\mathbf{s_0}$, or $\mathcal{A}_0^X$. $\mathcal{A}_0^X \subset \mathcal{A}_0^{\hat{I}}$ is more formally defined as

$$\mathbf{s}_{n+1} = \delta(\mathbf{s}_n, \mathbf{i}_n) \tag{3}$$

where $n \geq 0$ and $\mathbf{i}_n$ corresponds to x_n (remember that $\hat{I}$ and Σ are isomorphic and $X \in \Sigma^*$). And

$$\mathcal{A}_0^X = \{s_0, s_1, s_2, \dots, s_n\} \tag{4}$$

where n is the length of sequence X.

From the information about states gathered through the processing of the domain, a state machine of some kind, emulating the network, can be generated. The typically indeterministic data from the network must be processed in some way to lead to a deterministic discrete machine (e.g. [3]) or the extracted state machine can in itself be stochastic (e.g. [5]).

3 Experiments

The sets $\hat{\mathcal{A}}_0^X$ and $\hat{\mathcal{A}}_0^{X_B}$ are both subsets of $\mathcal{A}_0^{\hat{I}}$ but cover different aspect and generate different rule sets. In this paper we will experimentally investigate the relation between $\hat{\mathcal{A}}_0^X$ and $\hat{\mathcal{A}}_0^{X_B}$, i.e. the difference between the domain sampling and breadth first search approaches of RE in terms of the visited states.

In these experiments we have chosen to limit the tasks to be pure prediction tasks, i.e. the task for a network is to predict the next symbol in a sequence generated by a grammar and *not* to classify incoming strings. Another prerequisite is also that the networks are perfect, i.e. they never predict predictable symbols of the domain incorrectly. This in order to prevent illegal rules to be caused by an erroneous network, but instead to be indicators of flaws of the extraction procedure itself.

3.1 The Networks

Three prediction domains have been considered in this paper, two regular grammars and one context free. [10] showed that from an RNN effectively implementing a regular grammar, a finite state machine consistent with the RNN can be extracted. For the context free grammar, we assume that some limited version of it can be extracted.

- The simplest is the temporal XOR-problem, suggested in [9], where every third binary symbol is determined by an XOR operation of the two preceding symbols which are random.
- The next grammar, the "6-letter grammar", was created by Elman [9] to test a language with more than two symbols and that required some deeper memory in the network. The sequence from the grammar consists of the subsequences **ba**, **dii** and **guuu** concatenated in random order[1]. Consequently, only the vowels are predictable.

[1] In our experiments we used 'one-hot' encoding to represent the symbols to the network, i.e. six bits were used of which each one encodes only one symbol. Elman used a quite different non-orthogonal encoding based on phonological properties of the letters.

- The third domain was $\mathbf{0}^n\mathbf{1}^n$, a context free language. n was in these experiments $1 \leq n \leq 10$ and varied in random order with the generated strings concatenated into a single sequence. In this language, only the $\mathbf{1}$'s and the first $\mathbf{0}$ is predictable. The full grammar, with n unlimited, cannot be represented in any finite state machine, but since we only require the rules to correctly predict the training set it is possible to view this as a regular grammar (although this may be complicated if the network has actually learned to generalize to longer sequences).

These domains were chosen to test the effect of the number of symbols and language class separately. All languages have predictable and unpredictable parts of the generated sequences and the networks are all trained to predict the next symbol. 100 networks were trained on each domain until they were deemed to perfectly predict the predictable parts of the sequences. The architecture chosen was simple recurrent networks (SRNs) [9] with two hidden nodes. For the regular language backpropagation through time (BPTT) was used to train them and since BPTT had problems on the context free language an evolutionary hill-climbing algorithm was used for that instead.

3.2 Evaluation criteria

The primary objective of the experiments was to assert the degree of excess computational power used by rule extraction through breadth first in the selected domains. For all networks, we tested RE through breadth first search and sampling for varying values of q to see the effects of the quantization level on various aspects.

We chose to measure $|\hat{\mathcal{A}}_0^{X_B}|$ and $|\hat{\mathcal{A}}_0^{X}|$ and will here present the ratio, $|\hat{\mathcal{A}}_0^{X_B}|/|\hat{\mathcal{A}}_0^{X}|$, i.e. the relative difference in number of bins visited through RE and through processing of domain respectively. Also, the proportion of substrings in X_B that are at all possible in the domain which the network is trained on is measured. If the breadth first RE for example tests the sequence **00011110** on a correctly predicting $\mathbf{0}^n\mathbf{1}^n$-network starting from the initial state in the network, it is a symbol-sequence that never occurs in the true domain and should therefore be considered an obsolete sequence.

The performance of the extracted machines was also monitored to determine whether correct rules were extracted. The termination point for the breadth first RE, i.e. when the extracted machine is consistent with the data, was also tested in order to see if and when the algorithm would terminate.

4 Results

In Figure 1 we show an example of how RE can be illustrated in the state space of the RNN predicting the 6-letter sequence. In this example it can be seen how RE through breadth first search finds many states irrelevant for predicting within the domain.

In Figure 2 the ratios of visited bins and of syntactical substrings generated in the RNN by breadth first search RE in comparison to domain interaction are shown. It is clear that breadth first RE generated the biggest amount of irrelevant tests on the 6-letter networks. This is probably due to the fact that after each symbol in the 6-letter sequence, typically only one of six symbols will occur in the domain whereas all six symbols will be tested by the RE.

It should be mentioned that the RE algorithm terminated quite rapidly; for XOR within $q = 3$ to $q = 10$, for the 6-letter grammar within $q = 2$ to $q = 8$. But for $\mathbf{0}^n\mathbf{1}^n$ at least $q = 21$ and for 15% of the networks, the algorithm did not terminate at all. 84% of the XOR networks seemed to stabilize in terms of extracting equivalent machines. Only 2% of the 6-letter sequence stabilized. 5% of the $\mathbf{0}^n\mathbf{1}^n$ actually also stabilized. These numbers

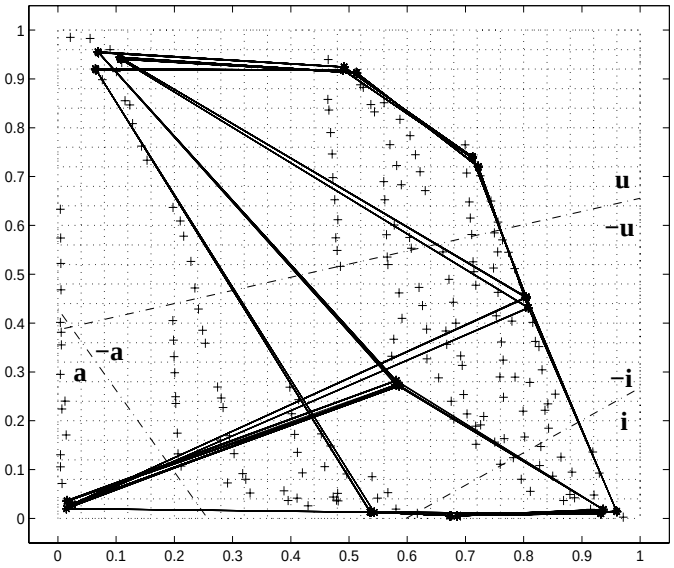

Figure 1: The internal activation of a network performing prediction in the 6-letter sequence. The lattice corresponds to the discretization with the state divided into 25^2 bins in this example. The diagonal dotted lines are the hyperplanes, defining the borders within the state space for which symbol that is predicted. The hyperplanes divides the state space into the '**u**'-region on the upper half, the '**a**'-region on the lower left side and the '**i**'-region on the lower right side. The rest of the state space corresponds to no valid symbol; the center area with all output nodes set to zero and a small area on the center left side with the '**a**' and '**u**'-node active simultaneously. The states visited through the breadth first RE are denoted '$+$' and the states visited through processing of the domain are denoted '$*$' and are connected to show the order of the states visited.

are not fully certain however, since the number of states in the minimized automata could continue increasing for higher quantization levels.

5 Discussion and Conclusions

We have shown that the degree to which breadth first RE requires excessive computational resources seems to be related to the number of symbols in the language for networks trained to predict symbolic sequences. The ratio of, for the domain, relevant "questions" (in form of sequences) "asked" to the network also was very low for the grammar with six symbols, and for the context free grammar.

Blair and Pollack [4] suggested to use the state count of the extracted machine to determine whether the network is effectively implementing "regular" or "non-regular" automaton. If the state count is growing indefinitely with q, they proposed to use this as an indicator that the underlying RNN is non-regular. But the results presented here suggest that, for prediction tasks, regularity of the network can not be tested as suggested in [4] since the number of states generated from networks predicting sequences of the regular languages was almost always growing indefinitely although the networks were predicting all symbols of the language perfectly. The percentage of networks for which the RE stabilized did also not correlate with the language class. The termination criterion of the RE algorithm was however satisfied much earlier for regular than for context free prediction networks. But

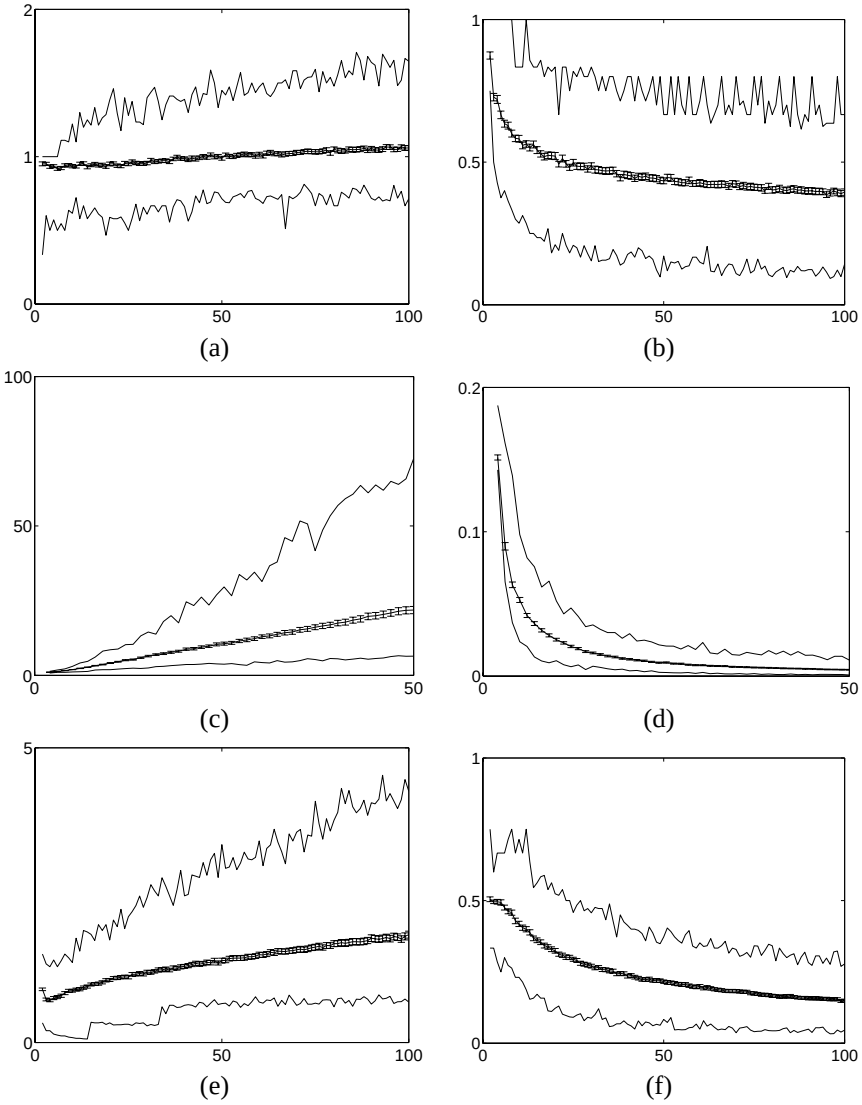

Figure 2: The ratio $|\hat{\mathcal{A}}_0^{X_B}|/|\hat{\mathcal{A}}_0^{X}|$ is shown in the left column and the ratio of substrings in X_B possible in the domain is shown on the left side. (a) and (b) correspond to the XOR-language, (c) and (d) to the 6-letter language (observe that for this language q was at most 50) and (e) and (f) to the $\mathbf{0}^n\mathbf{1}^n$-language. The maximum, minimum, average and standard deviation of one hundred networks for each domain are shown.

this could also be due to the larger number of states needed to model the strictly *regular language* $\mathbf{0}^n\mathbf{1}^n$ with $1 \leq n \leq 10$. This should however be investigated further to give more insight into if (and how) RE can be used to determine the underlying language class, which is judged to be "fool's gold" by Kolen [11].

One can also argue that RE through search is, in some sense, less credible than through sampling since it requires the possibility of an external entity *setting* the state of the network. Sampling of the networks internal states generated in the context of its domain how-

ever generates stochastic machines that are harder to analyse (and to minimize, execute, compare etc.) than the finite automata generated by breadth first search RE.

We suggest that in most "real world domains", e.g. stock market prediction, the task is precisely to *predict* sequences of data with typically a magnitude of possible input patterns. According to our results, in these types of tasks it would be especially beneficial to use sampling rather than breadth first to extract rules.

But, to fully exploit the potential of RE through sampling and to ensure further development of these algorithms, new questions need to be asked. For example, the optimal quantization function for the state space should be sought. And to do that, we need to ask how to evaluate different quantization functions. Since an RNN (as defined here) is deterministic, one possibility could be to give higher scores to quantization functions generating "less stochastic" machines. Another difficulty that has not been investigated properly is how rule extraction from imperfect networks (very commonly found in real world domains) should be conducted. In a way, this has been implicitly touched in this paper, since we used partly unpredictable prediction domains, but this should be investigated in further detail.

References

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

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

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

[4] A. Blair and J. Pollack. Analysis of dynamical recognizers. *Neural Computation*, 9(5):1127–1142, 1997.

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

[6] A. B. Tickle and R. Andrews. The truth will come to light: directions and challenges in extracting the knowledge embedded within mined artificial neural networks. *IEEE Trans. on Neural Networks*, 9(6):1057–1068, 1998.

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

[8] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Compilation*. Addison-Wesley Publishing Company, 1979.

[9] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

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

[11] J. F. Kolen. Fool's gold: Extracting finite state machines from recurrent network dynamics. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Neural Information Processing Systems 6*, pages 501–508, San Francisco, CA, 1993. Morgan Kaufmann.