Analysis of Dynamical Recognizers

Alan D. Blair & Jordan B. Pollack
Dept. of Computer Science
Volen Center for Complex Systems
Brandeis University
Waltham, MA 02254-9110
blair@cs.brandeis.edu pollack@cs.brandeis.edu

September 7, 1995
(revised June 14, 1996)

Abstract

Pollack (1991) demonstrated that second-order recurrent neural networks can act as dynamical recognizers for formal languages when trained on positive and negative examples, and observed both phase transitions in learning and IFS-like fractal state sets. Follow-on work focused mainly on the extraction and minimization of a finite state automaton (FSA) from the trained network. However, such networks are capable of inducing languages which are not regular, and therefore not equivalent to any FSA. Indeed, it may be simpler for a small network to fit its training data by inducing such a non-regular language. But when is the network's language **not** regular? In this paper, using a low dimensional network capable of learning all the Tomita data sets, we present an empirical method for testing whether the language induced by the network is regular or not. We also provide a detailed ε-machine analysis of trained networks for both regular and non-regular languages.

1 Introduction

Pollack (1991) showed one way a recurrent network may be trained to recognize formal languages from examples. The resulting networks often displayed complex limit dynamics which were fractal in nature (Kolen, 1993).

Alternative architectures had been employed earlier for related tasks (Jordan, 1986, Pollack, 1987, Cleeremans et al., 1989). Others have been proposed since (Watrous & Kuhn, 1992, Frasconi et al., 1992, Zeng et al., 1994, Das & Mozer, 1994, Forcada & Carrasco, 1995) and a number of approaches to analysing recurrent networks have been developed. One of the principal themes has been the use of clustering and minimization techniques to extract a Finite State Automaton (FSA) that approximates the network's behavior (Giles et al., 1992, Manolios & Fanelli, 1994, Tiňo & Sajda, 1995).

Casey (1996) showed that if the network robustly models an FSA, the method proposed in (Giles et al., 1992) will successfully extract the FSA given a fine enough resolution. However in many cases the language induced by the network is not regular, and therefore cannot be modeled exactly by any FSA. In order to analyse the behavior of these networks, we need a method for testing empirically whether the network's behavior is regular (i.e. it is modeling and FSA) or not.

In the present work, we show how to perform an analysis of the trained network at high resolution by constructing a series of finite state approximations which describe its behavior at progressively more refined levels of detail. With this analysis we are able to measure empirically when the network has crossed over the boundary between regular and non-regular behavior.

2 Architecture

Our system is comprised of a gated recurrent neural network with two subnetworks W_0 & W_1, a gating device and a perceptron P. For $\sigma = 0, 1$ we denote by W_σ^{jk} the real-valued weights of subnetwork W_σ and by w_σ the function it implements.

To operate the network, we first feed the initial vector x_0 into the input layer. Then, as each symbol of a binary string $\Sigma = \sigma_1 \ldots \sigma_n$ is read in, a gating device chooses one of the two subnetworks W_0 or W_1, depending on whether the next symbol σ_t is equal to 0 or 1. The current real-valued hidden unit state vector x_{t-1} is then fed through this subnetwork W_{σ_t} to produce the next state $x_t = w_{\sigma_t}(x_{t-1})$ given by

$$x_t^j = \tanh\left(\mathrm{W}_{\sigma_t}^{j0} + \sum_{k=1}^{d} \mathrm{W}_{\sigma_t}^{jk} x_{t-1}^k\right), \qquad \text{for} \quad 1 \le j \le d, \quad 1 \le t \le n.$$

This part of the architecture is equivalent to the second order recurrent networks used in (Pollack, 1991) and (Giles et al., 1992), with a slightly different notation.

After the whole string has been read, the final state x_n is fed through a perceptron P producing output

$$z = \tanh\left(\mathrm{P}_0 + \sum_{j=1}^{d} \mathrm{P}_j x_n^j\right),$$

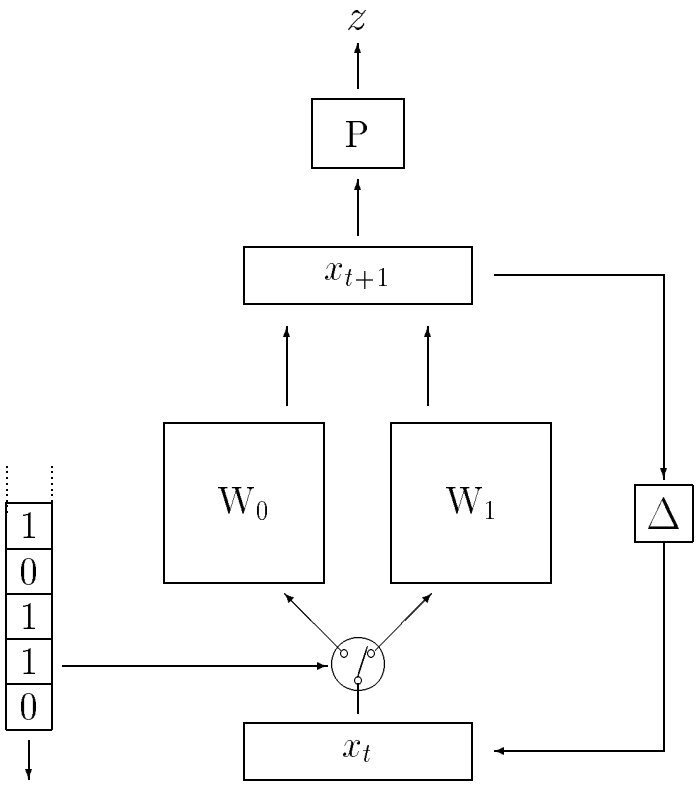


Figure 1: System architecture

where each P_j is a real-valued weight and z is the real-valued output.

To make the geometry more accessible, we use the interval $[-1, 1]$ for network activations and adopt the hyperbolic tangent as our transfer function, which is equivalent by rescaling to the usual sigmoid function. The object of training is that the output z should be close to $+1$ for accept strings and close to -1 for reject strings ('close' meaning within, say, 0.4). The class of languages recognizable by such networks lies somewhere between the regular and recursive language families, and is known to contain some languages that are not context-free (Siegelmann & Sontag, 1992). In the work reported here the state-space dimension d was always equal to 2, which gives the model a total of 17 free parameters – 3 for the perceptron, 6 for each subnetwork and 2 for the initial point. As outlined below, this architecture was adequate to the task of learning any of the data sets from (Tomita, 1982) within a few hundred training epochs.

3 Analysis

Our purpose here is not only to train the network but also to gain some insight into how it accomplishes its assigned task [see (Casey, 1996) for another, related approach to this problem]. If the recurrent layer has d nodes, the state space of the system is the d-dimensional hypercube $\mathcal{X} = [-1, 1]^d$. However we need not be concerned with the whole state space, but only with that portion of it – which we call $\mathcal{A}_0$ – that is accessible from the initial point x_0 via repeated applications of the maps w_0 & w_1. Formally, we may define $\mathcal{A}_0$ thus:

$$\begin{array}{ll} \text{(i)} & \mathcal{Y}_0 = \{x_0\} \\ \text{(ii)} & \text{For } i \geq 0, \quad \mathcal{Y}_{i+1} = \mathcal{Y}_i \cup w_0(\mathcal{Y}_i) \cup w_1(\mathcal{Y}_i) \quad [\text{Note: } \mathcal{Y}_{i+1} \supseteq \mathcal{Y}_i] \\ \text{(iii)} & \mathcal{A}_0 = \lim_{i \to \infty} \mathcal{Y}_i \end{array} \tag{3.1}$$

Once we have identified the space $\mathcal{A}_0$ on which the dynamics take place, the next step is to find an appropriate *subdivision* of this space – i.e. a finite collection of disjoint nonempty subsets which cover $\mathcal{A}_0$ – that is compatible with the maps w_0 & w_1. The coarsest possible subdivision is provided by the perceptron, which divides $\mathcal{A}_0$ into the subset $\mathcal{U}_{\text{acc}}$ of 'accept' points and the subset $\mathcal{U}_{\text{rej}}$ of 'reject' points. So we take as our first subdivision $\mathcal{S}_1 = \{\mathcal{U}_{\text{acc}}, \mathcal{U}_{\text{rej}}\}$. If $\mathcal{S} = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\}$ is a subdivision, the *pre-image* of $\mathcal{S}$ under w_0 is

$$w_0^{-1}(\mathcal{S}) = \{w_0^{-1}\mathcal{U}_1, \ldots, w_0^{-1}\mathcal{U}_m\} \setminus \{\emptyset\},$$

where

$$w_0^{-1}\mathcal{U}_j = \{x \in \mathcal{A}_0 \,|\, w_0(x) \in \mathcal{U}_j\}$$

[and similarly for $w_1^{-1}(\mathcal{S})$]. The *join* $\mathcal{R} \vee \mathcal{S}$ of two partitions $\mathcal{R}$ and $\mathcal{S}$ is

$$\mathcal{R} \vee \mathcal{S} = \{\mathcal{U} \cap \mathcal{V} \,|\, \mathcal{U} \in \mathcal{R}, \mathcal{V} \in \mathcal{S}\} \setminus \{\emptyset\}.$$

Using these tools, we define a series of subdivisions $\{\mathcal{S}_i\}_{i \geq 1}$ by

$$\begin{array}{ll} \text{(i)} & \mathcal{S}_1 = \{\mathcal{U}_{\text{acc}}, \mathcal{U}_{\text{rej}}\} \\ \text{(ii)} & \text{For } i \geq 1, \ \mathcal{S}_{i+1} = \mathcal{S}_i \vee w_0^{-1}(\mathcal{S}_i) \vee w_1^{-1}(\mathcal{S}_i) \end{array}$$

We are now ready to construct a series of non-deterministic FSA's $\{\mathrm{M}_i\}_{i \geq 1}$ which approximate the behavior of the network. For each i, let M_i be as follows:

(1) the states of M_i are indexed by the elements $\mathcal{U}_j$ in the subdivision $\mathcal{S}_i$
(2) the initial state is that $\mathcal{U}_j$ which contains x_0
(3) the accept states are those $\mathcal{U}_j$ contained in $\mathcal{U}_{\text{acc}}$
(4) an edge labeled by 0 connects $\mathcal{U}_j$ to $\mathcal{U}_k$ exactly when $\mathcal{U}_j \cap w_0^{-1}\mathcal{U}_k \neq \emptyset$
(5) an edge labeled by 1 connects $\mathcal{U}_j$ to $\mathcal{U}_k$ exactly when $\mathcal{U}_j \cap w_1^{-1}\mathcal{U}_k \neq \emptyset$

Each FSA in the series is a refinement of the previous one which sketches out some of the details of its non-determinism.

Proposition. *If M_i is deterministic for some i then M_j will equal M_i for all $j > i$ and M_i will exactly model the network's behavior.*

Proof: Suppose M_i is deterministic for some i and consider the input string $\Sigma = \sigma_1 \ldots \sigma_n$, with $\sigma_j \in \{0, 1\}$ for $1 \leq j \leq n$. We must show that Σ is accepted by M_i if and only if it is accepted by the network. Recall that x_0 denotes the initial point and let $x_j = w_{\sigma_j}(x_{j-1})$, for $1 \leq j \leq n$. Recall that the states of M_i are indexed by the elements $\mathcal{U}_k$ of subdivision $\mathcal{S}_i$ and for $0 \leq j \leq n$ let $\mathcal{U}^{(j)}$ be that subset $\mathcal{U}_k$ which contains x_j. Then in particular $\mathcal{U}^{(0)}$ contains x_0 and is therefore the initial state of M_i. Since $x_j = w_{\sigma_j}(x_{j-1})$ we have

$$\mathcal{U}^{(j-1)} \cap w_{\sigma_j}^{-1}\mathcal{U}^{(j)} \neq \emptyset,$$

so $\mathcal{U}^{(j-1)}$ is connected to $\mathcal{U}^{(j)}$ with an edge labeled by σ_j. As M_i is deterministic, this must be the only such edge emanating from $\mathcal{U}^{(j-1)}$. By induction on j it follows that M_i must be put into state $\mathcal{U}^{(j)}$ when string $\sigma_1 \ldots \sigma_j$ is input. In particular, once the whole string has been input, M_i will be in state $\mathcal{U}^{(n)}$, so

$$\begin{aligned} \Sigma \text{ is accepted by } \mathrm{M}_i &\Leftrightarrow \mathcal{U}^{(n)} \subseteq \mathcal{U}_{\mathrm{acc}} \\ &\Leftrightarrow x_n \in \mathcal{U}_{\mathrm{acc}} \\ &\Leftrightarrow \Sigma \text{ is accepted by the network.} \end{aligned}$$

To show that $\mathrm{M}_{i+1} = \mathrm{M}_i$ and no further subdivisions are made, suppose to the contrary that there are two points x_1 and x_2 which lie in the same element $\mathcal{U}_j$ of the partition $\mathcal{S}_i$, but in different elements of $\mathcal{S}_{i+1} = \mathcal{S}_i \vee w_0^{-1}(\mathcal{S}_i) \vee w_1^{-1}(\mathcal{S}_i)$. Then it must be for $\sigma =$ either 0 or 1 that $w_\sigma(x_1)$ lies in some $\mathcal{U}_k \in \mathcal{S}_i$ while $w_\sigma(x_2)$ lies in $\mathcal{U}_l$ with $\mathcal{U}_k \neq \mathcal{U}_l$. But then $\mathcal{U}_j$ would be connected to both $\mathcal{U}_k$ and $\mathcal{U}_l$ with edges labeled by σ, contradicting the assumption that M_i was deterministic. □

If the induced language is not regular, the machines M_i will in theory continue to grow indefinitely in complexity. Of course in practice we cannot implement the above procedure exactly for our trained networks, but must instead use a discrete approximation. Following the analysis of (Giles et al., 1992) we can approximate $\mathcal{A}_0$ computationally along the lines of (3.1) as follows: first 'discretize' the system at resolution r by dividing the state space into r^d points in a regular lattice and approximating w_0 & w_1 with functions $\hat{w}_0$ & $\hat{w}_1$ that act on these points such that $\hat{w}_\sigma(\hat{x})$ is the nearest lattice point to $w_\sigma(\hat{x})$. Each $\mathcal{Y}_i$ will then be represented by a finite set $\hat{\mathcal{Y}}_i$ and the condition $\hat{\mathcal{Y}}_{i+1} \supseteq \hat{\mathcal{Y}}_i$ guarantees that the procedure will terminate to produce a discrete approximation $\hat{\mathcal{A}}_0$ for $\mathcal{A}_0$.

In discrete form, the above procedure may be equated with the Hopcroft Minimization Algorithm (Hopcroft & Ullman, 1979), or the method of ε-machines (Crutchfield & Young, 1990, Crutchfield, 1994), and was first used in the present context by Giles et al. (1992). Using small values of r, their aim was to extract an FSA that, while it might not model the network's behavior exactly, would model it closely enough to faithfully classify the training data.

We had in mind a different purpose: namely to test empirically whether the network is regular (i.e. modeling an FSA) or not, and to analyse its behavior in more fine-grained detail. The minimization procedure described above is bound to terminate in finite time due to the finite resolution of the representation. However as the resolution is scaled up, the size of the largest FSA generated should in principle stabilize in the case of regular networks and grow rapidly in the case of non-regular ones. Therefore, as an empirical test of regularity, we perform these computations at two different resolutions (200×200 and 500×500) and report (in rows 4 and 5 of Table 1) the size (i.e. number of states) of the largest FSA generated. If the largest FSA's generated at the different resolutions are the same, we take this as an indication that the network is regular; if they are vastly different, that it is not regular. This allows us to probe the nature of the network's behavior as the training progresses. In some cases the network undergoes a 'phase transition' to regularity at some point in its training and we are able to measure with considerable precision the time at which this phase transition occurs. It should be stressed, however, that these discretized

computations do not constitute a formal proof, but merely provide strong empirical evidence as to the regularity or otherwise of the network and its induced language. More rigorous results may be found in Casey (1996, 1993) who described the general dynamical properties that a neural network or any dynamical system has to have in order to robustly model an FSA. Tiňo et al. (1995) also made a detailed analysis of networks with a 2-dimensional state space under certain assumptions about the sign and magnitude of the weights.

It is important also to note that the regularity we are testing for is a property of the trained network and not an intrinsic property of the input strings, since the (necessarily finite) training data may be generalized in an infinite number of ways, each producing an equally valid induced language that may be regular or non-regular. Good symbolic algorithms exist already for finding a regular language compatible with given input data (Trakhtenbrot & Barzdin', 1973, Lang, 1992). Our purpose is rather to analyse the kinds of languages that a dynamical system such as a neural network is likely to come up with when trained on that data. We do not claim that our methods are efficient when scaled up to higher dimensions. Rather, it is hoped that a detailed analysis of networks in low dimensions will lead to a better understanding of the phenomenon of regular vs. non-regular network behavior in general.

Finally, we remark that the functions w_0 and w_1 map $\mathcal{X}$ continuously into itself and as such define an *Iterated Function System* or IFS (Barnsley, 1988) as noted in (Kolen, 1994). The *attractor* $\mathcal{A}$ of this IFS may be defined as follows:

(i) $\mathcal{Z}_0 = \mathcal{X}$
(ii) For $i \geq 0$, $\mathcal{Z}_{i+1} = w_0(\mathcal{Z}_i) \cup w_1(\mathcal{Z}_i)$ [by induction $\mathcal{Z}_{i+1} \subseteq \mathcal{Z}_i$]
(iii) $\mathcal{A} = \bigcap_{i \geq 0} \mathcal{Z}_i$

As with $\mathcal{A}_0$, we can find a discrete approximation $\hat{\mathcal{A}}$ for $\mathcal{A}$ at any given resolution. We have found empirically that the convergence to the attractor was very rapid for our trained networks so that $\hat{\mathcal{A}}_0$ was equal to a subset of $\hat{\mathcal{A}}$ plus a very small number of 'transient' points. For this reason we call $\mathcal{A}_0$ a *subattractor* of the IFS. If w_0 and w_1 were contractive maps, $\mathcal{A}_0$ would contain the whole of $\mathcal{A}$ (Barnsley, 1988), but in our case they are generally not contractive so $\hat{\mathcal{A}}_0$ may contain only part of $\hat{\mathcal{A}}$. The close relationship between $\hat{\mathcal{A}}_0$ and $\hat{\mathcal{A}}$ should make it possible to analyse the general family of languages accepted by the network as x_0 varies, though we do not pursue this line of enquiry here.

4 Results

Networks with the architecture described in §2 were trained to recognize formal languages using backpropagation through time (Williams & Zipser, 1989, Rumelhart et al., 1986), with a modification similar to Quickprop (Fahlman, 1989)[1] and a learning rate of 0.03. The weights were updated in batch mode, and the perceptron weights P_j

[1]Specifically, the cost function we used was

$$E = -\frac{1}{2}(1+s)^2 \log(\frac{1+z}{1+s}) - \frac{1}{2}(1-s)^2 \log(\frac{1-z}{1-s}) + s\,(z-s)$$

Table 1: Epochs to Learning (L) and Regularity (R) for the seven Tomita languages.

Network	N1	N2	N3	N4		N5	N6		N7
	L/R	L	L	L	R	L/R	L	R	L
epochs to Learning	200	600	400	200		800	200		600
epochs to Regularity	200	–	–		400	800		600	–
200×200 FSA size	2	341	272	2052	4	21	7123	3	264
500×500 FSA size	2	881	808	7248	4	7	39200	3	806
Tomita's FSA size	2	3	5		4	4		3	5

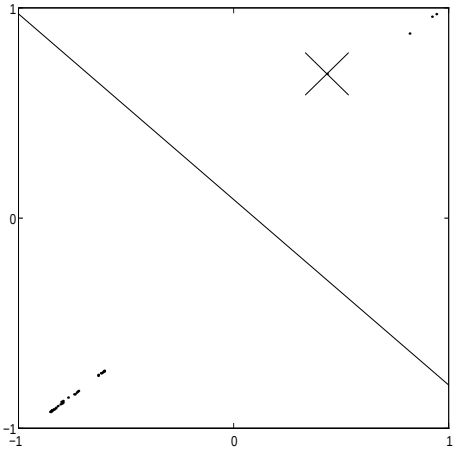
were constrained by rescaling to the region where $\sum_{j=1}^{d} \mathrm{P}_j{}^2 \leq 1$. In contrast to (Pollack, 1991) where the backpropagation was truncated, we backpropagated through all levels of recurrence as in (Williams & Zipser, 1989). In addition, we allowed the initial point x_0 to vary as part of the backpropagation – a modification that was also developed in independent work by Forcada & Carrasco (1995).

Our seven groups of training strings (see Appendix) were copied exactly from (Tomita, 1982) except that we did not include the empty string in our training sets. Rows 4 and 5 of Table 1 show the size (i.e. number of states) of the largest FSA generated from the trained networks at two different resolutions by the methods described in §3. For comparison, row 6 shows the size of the minimal FSA's which Tomita found by exhaustive search. A network can be said to have learned all its training data once the output is positive for all accept strings and negative for all reject strings (i.e. a maximum error of 1.0), but it makes sense to aim for a lower max error in order to provide a 'safety margin'. For network N5, the max error reached its lowest value of 0.46 after 800 epochs. The other networks were trained until they achieved a max error of less than 0.4, the number of epochs required being shown in row 2 (labeled 'epochs to Learning'). Since the test for regularity is computationally intensive, we did not test our networks at every epoch but only at intervals of 200 epochs.

For network N1 the derived FSA is the same for both resolutions, providing evidence that the induced language is regular. For networks N2, N3, N4, N6 and N7 on the other hand, the max FSA grows dramatically as we increase the resolution, suggesting that the induced language is not regular. We continued to train these networks to see if they would later become regular, and found that networks N4 and N6 became regular after 400, 600 epochs, respectively, as indicated in row 3 (labeled 'epochs to Regularity'), while networks N2, N3 and N7 remained non-regular even after 10,000 epochs. For N5 the size of the max FSA actually decreased with higher resolution from 21 to 7, suggesting that the induced language is regular but that high resolution is required to detect its regularity.

(where z is the actual output and s the desired output) which leads to a 'delta rule' of

$$\delta = (1 - sz)(s - z).$$



$$W_0 = \begin{bmatrix} -0.89 & -0.09 & -0.14 \\ -1.13 & -0.09 & -0.14 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 0.20 & 0.68 & 0.96 \\ 0.20 & 0.81 & 1.19 \end{bmatrix}$$

$$P = \begin{bmatrix} -0.07 & 0.66 & 0.75 \end{bmatrix}$$

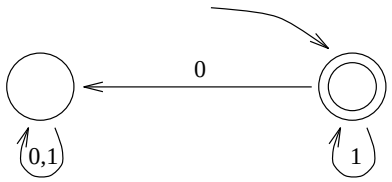
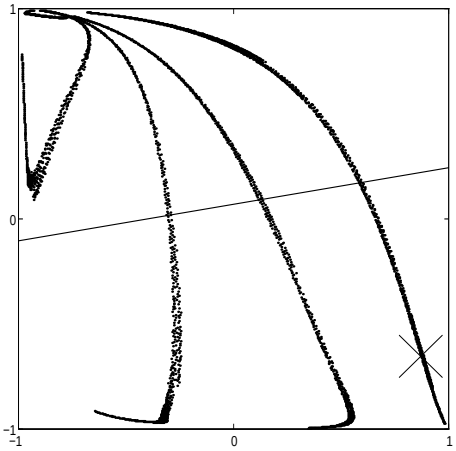
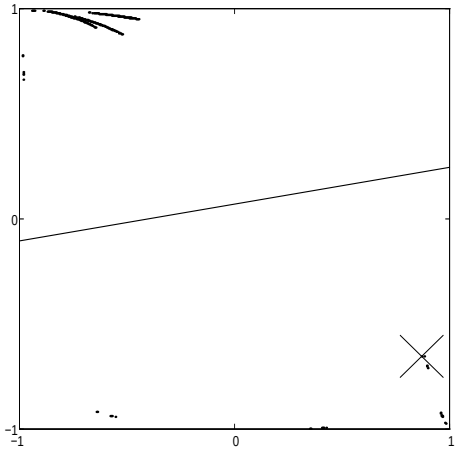


Figure 2: Subattractor, weights and equivalent FSA for network N1.





$$W_0 = \begin{bmatrix} -0.567 & 1.761 & 0.815 \\ -0.219 & -2.591 & 0.446 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 0.752 & 0.548 & -1.071 \\ 0.074 & -0.813 & 1.502 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.069 & 0.172 & -0.985 \end{bmatrix}$$

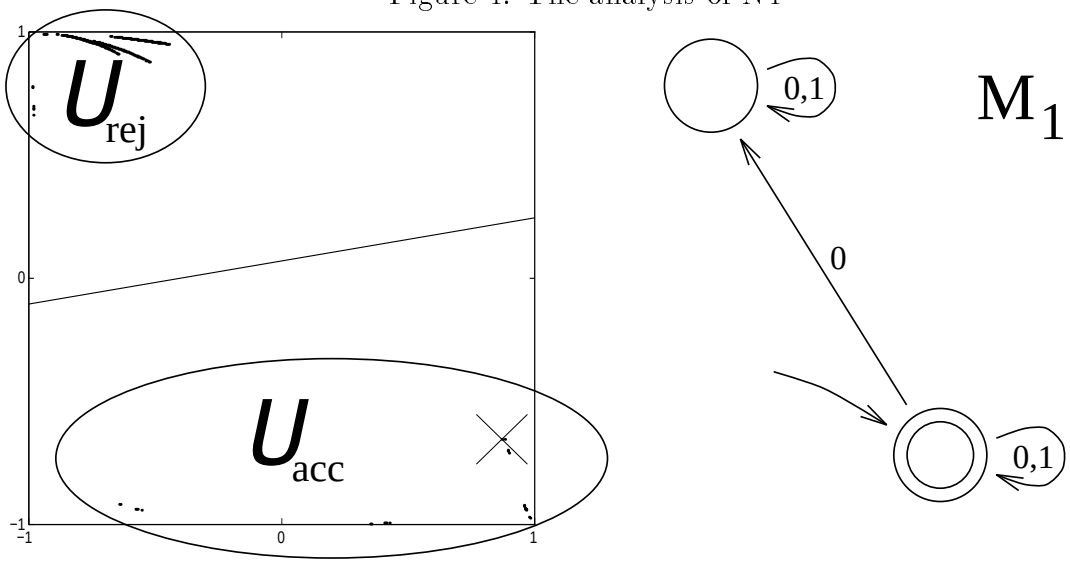
$$W_0 = \begin{bmatrix} -0.567 & 1.763 & 0.816 \\ -0.219 & -2.593 & 0.446 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 0.751 & 0.549 & -1.073 \\ 0.075 & -0.813 & 1.502 \end{bmatrix}$$

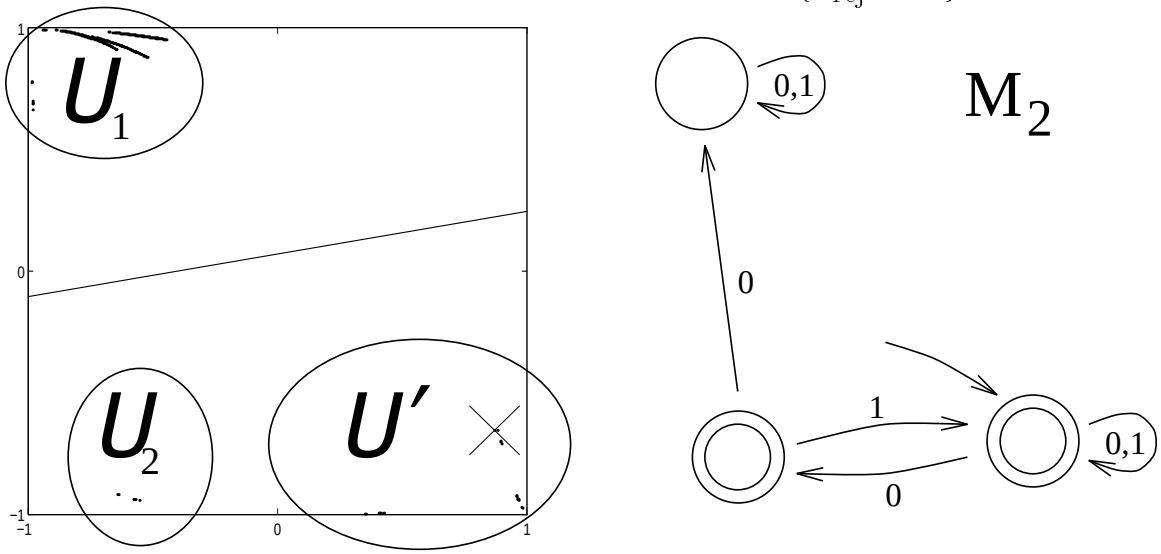
$$P = \begin{bmatrix} 0.069 & 0.173 & -0.985 \end{bmatrix}$$

Figure 3: The phase transition of N4.

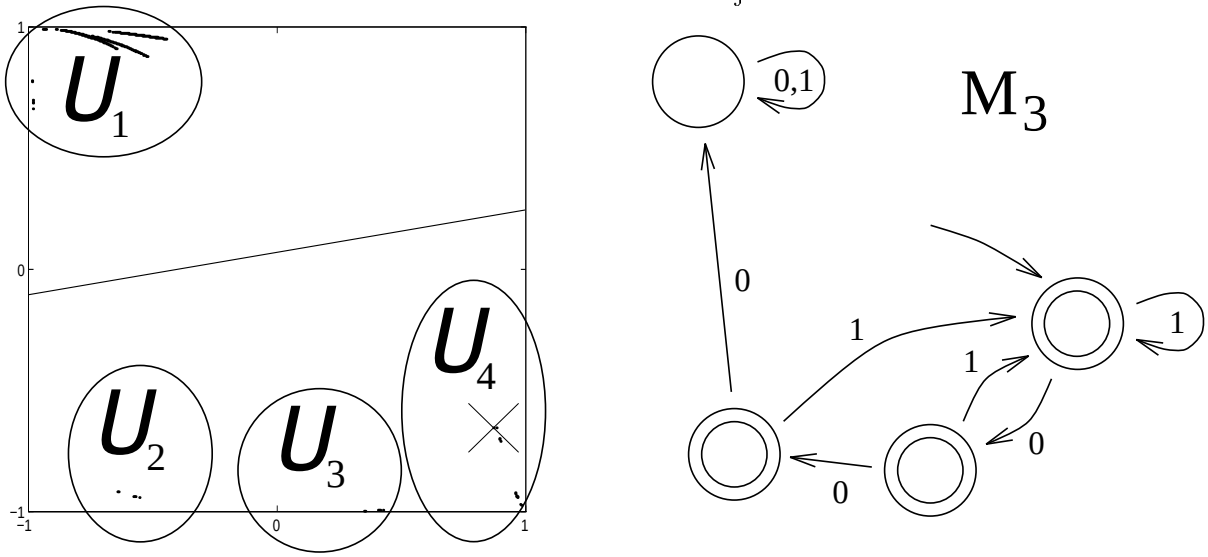
Figure 4: The analysis of N4



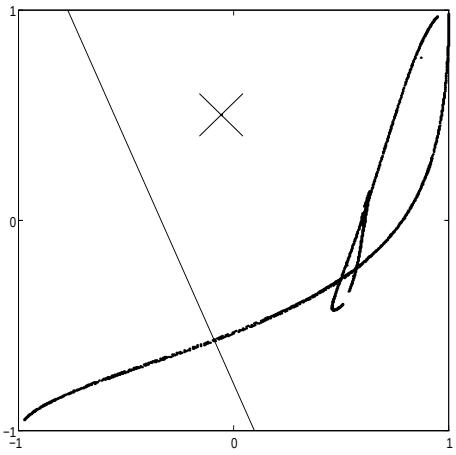
Step 1: the perceptron line imposes the subdivision $\mathcal{S}_1 = \{\mathcal{U}_{\rm rej}, \mathcal{U}_{\rm acc}\}$



Step 2: $\mathcal{S}_2 = \{\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}'\}$, where $\mathcal{U}_2 = \mathcal{U}_{\rm acc} \cap w_0^{-1}\mathcal{U}_{\rm rej}$, $\mathcal{U}' = w_0^{-1}\mathcal{U}_{\rm acc}$



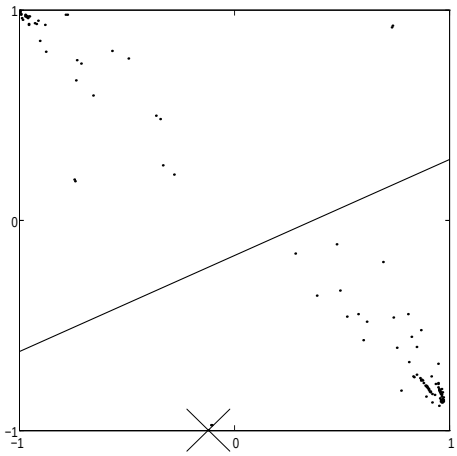
Step 3: $\mathcal{S}_3 = \{\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4\}$, where $\mathcal{U}_3 = w_0^{-1}\mathcal{U}_2$, $\mathcal{U}_4 = w_0^{-1}\mathcal{U}'$
M_3 is deterministic, so no further subdivisions are made.



$$W_0 = \begin{bmatrix} 1.51 & -1.71 & -2.03 \\ 0.29 & -1.01 & -1.12 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 1.19 & -0.81 & 0.18 \\ 0.86 & -1.45 & 0.16 \end{bmatrix}$$

$$P = \begin{bmatrix} -0.31 & -0.92 & -0.40 \end{bmatrix}$$



$$W_0 = \begin{bmatrix} 0.02 & 1.16 & -1.06 \\ 0.48 & -2.58 & -0.85 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} -1.74 & -1.35 & 1.92 \\ 1.41 & 0.69 & -1.87 \end{bmatrix}$$

$$P = \begin{bmatrix} -0.15 & 0.42 & -0.91 \end{bmatrix}$$

Figure 5: N2, which is not regular, and N5, which is regular but with no obvious clusters.

As an illustration, Figure 2 shows the subattractor for N1, which learned its training data after 200 epochs. The axes measure the activations of hidden nodes x^1 and x^2, respectively. The initial point x_0 is indicated by a cross (upper right corner). Also shown is the dividing line between accept points and reject points. Roughly speaking, w_0 squashes the entire state space into the lower left corner, while w_1 flattens it out onto the line $x_1 = x_2$. The dividing line separates $\mathcal{A}_0$ into two pieces - $\mathcal{U}_{\text{acc}}$ in the upper right corner and $\mathcal{U}_{\text{rej}}$ in the lower left. w_1 maps each piece back into itself, while w_0 maps both pieces into $\mathcal{U}_{\text{rej}}$. The ε-machine method thus produces the 2-state FSA shown at right. (Note: final states are indicated by a double circle, the initial state by an open arrow). In this simple case the FSA can be shown to exactly model the network's behavior. Of course N1 was trained on an extremely easy task that could have been learned with even fewer weights.

Figure 3 shows network N4 captured at its phase transition. After 371 epochs (left) it has learned the training set, but the induced language is not regular. At the next epoch (right) it has become regular. Though the weights have been shifted by only 0.004 units in euclidean space, the dynamics of the two networks are quite different. Applied to the left network at 500×500 resolution, our analysis produced a series of FSA's of maximum size 2564. Applied to the right network, it terminated after three steps to produce the same 4-state FSA that Tomita (1982) found by exhaustive search. The states of the FSA correspond to the following four subsets of $\mathcal{A}_0$: $\mathcal{U}_1$ in the upper left corner, $\mathcal{U}_2$ around $(-0.6, -0.9)$, $\mathcal{U}_3$ barely visible at $(0.4, -1.0)$ and $\mathcal{U}_4$ in the lower right corner. The details of the successive subdivisions are outlined in Figure 4.

The above examples would also be amenable to previous, clustering-based, approaches because of the way they 'partition [their] state space into fairly well-separated, distinct regions or clusters' as hypothesized in (Giles et al., 1992). Those shown in Figure 5 seem to be trickier. The subattractor for N5 (right) appears to be a bunch of scattered points with no obvious clustering, yet our fine-grained analysis was able to extract from it a 7-node FSA – a little larger than the minimal FSA of 4 nodes found by Tomita.

Network N2 (left) seems to have induced a non-regular language. Figure 6 shows the first four iterations of analysis applied to it. Note that each FSA refines the previous one bringing to light more details. Much can be learned about the induced language by examining these finite state approximations. For example, we can infer from M_2 that the network rejects all strings ending in 1, from M_3 that it accepts all nonempty strings of the form $(10)^*$, but rejects any string ending in 110.

5 Conclusion

By allowing the decision boundary and the initial point to vary, our networks with two hidden nodes were able to induce languages from all the data sets of (Tomita, 1982) within a few hundred epochs.

Many researchers implicitly regard an extracted FSA as superior to the trained network from which it was extracted (Omlin & Giles, 1996) with regard to predictability, compactness of description, and to the particular way each of them 'generalizes' to classify new, unseen input strings. For this reason, earlier work in the field had focused on extracting an FSA which approximates the behavior of the network. However, that approach is imprecise if the network has induced a non-regular language and does not exactly model an FSA. We have provided a fine-grained analysis for a number of trained networks, both regular and non-regular, using an approach similar to the method of ε-machines which Crutchfield & Young (1990) used to analyse certain hand-crafted dynamical systems. In particular, we were able to measure empirically whether the induced language was regular or not.

The fact that several of the networks induced non-regular languages suggests a discrepancy between languages which are "simple" for dynamical recognizers and those which are "simple" from the point of view of automata theory (namely, the regular languages). It is easier for these learning systems to induce a non-regular language to fit the sparse data of the Tomita training sets, rather than the expected minimal regular language. The use of comprehensive training sets, or intentional heuristics, might constrain networks away from these interesting dynamics.

It could be argued that the network and FSA ought to be seen on a more equal footing, since the 17 parameters of the network provide a compactness of description comparable to that of the FSA, and the language induced by the network is in principle on a par with that of the FSA in the sense that they both generalize the same training data. We hope that further work in this direction may lead to a better understanding of network dynamics and help to clarify, compare and contrast the relative merits of symbolic and dynamical systems.

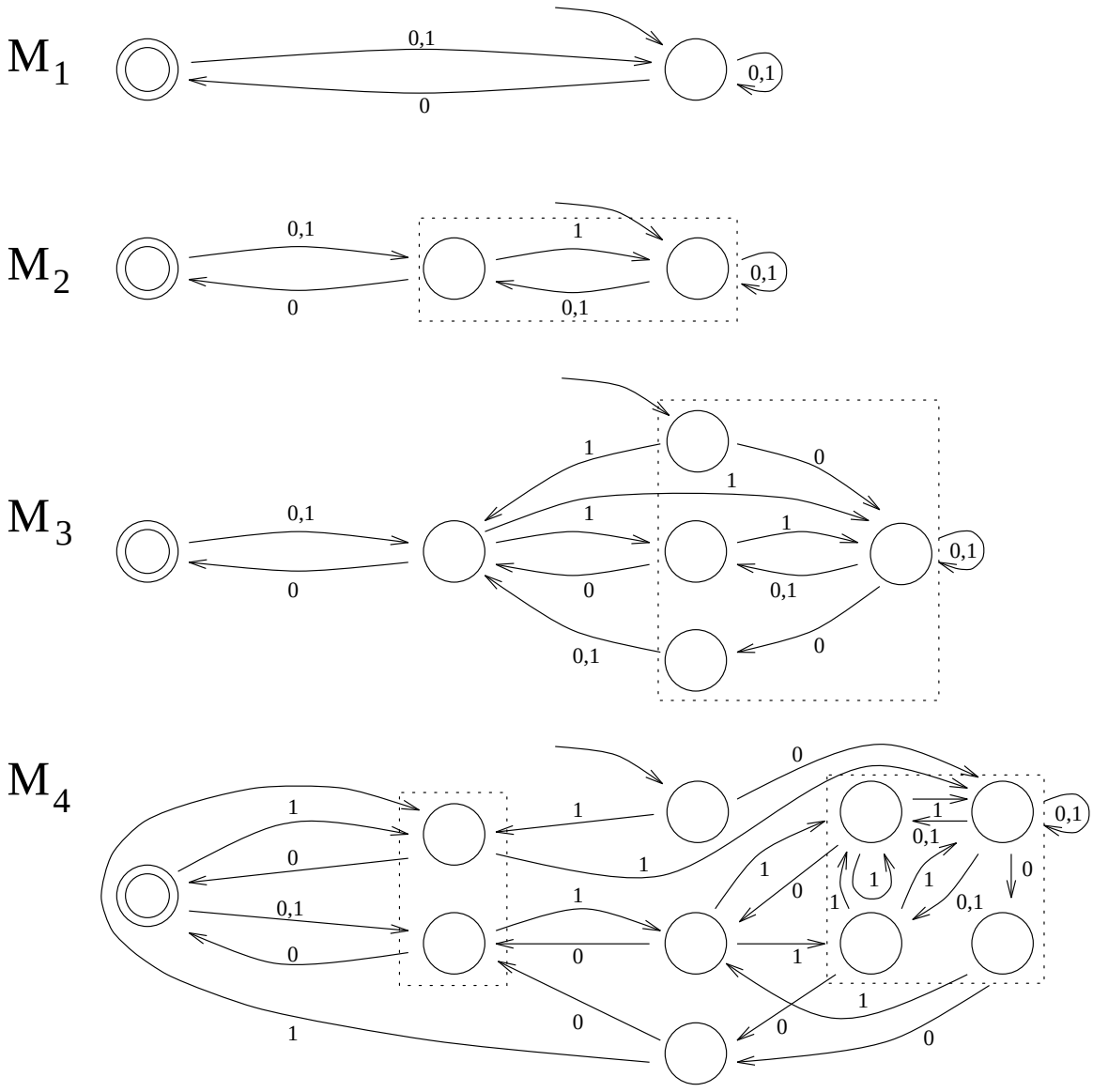


Figure 6: The first four steps of analysis applied to N2.

6 Acknowledgments

This research was funded by a Krasnow Foundation Postdoctoral Fellowship, by ONR grant N00014-95-0173 and by NSF grant IRI-95-29298. We are indebted to David Wittenberg for helping to improve its presentation, and to Mike Casey for stimulating discussions.

7 Appendix: Tomita's Data Sets

N1 Accept

```
1
1 1
1 1 1
1 1 1 1
1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
```

N1 Reject

```
0
1 0
0 1
0 0
0 1 1
1 1 0
1 1 1 1 1 1 1 0
1 0 1 1 1 1 1 1
```

N2 Accept

```
1 0
1 0 1 0
1 0 1 0 1 0
1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0
```

N2 Reject

```
1
0
1 1
0 0
0 1
1 0 1
1 0 0
1 0 0 1 0 1 0
1 0 1 1 0
1 1 0 1 0 1 0 1 0
```

N3 Accept

```
1
0
0 1
1 1
0 0
1 0 0
1 1 0
1 1 1
0 0 0
1 0 0 1 0 0
1 1 0 0 0 0 0 1 1 1 0 0 0 0 1
1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 0 0
```

N3 Reject

```
1 0
1 0 1
0 1 0
1 0 1 0
1 1 1 0
1 0 1 1
1 0 0 0 1
1 1 1 0 1 0
1 0 0 1 0 0 0
1 1 1 1 1 0 0 0
0 1 1 1 0 0 1 1 0 1
1 1 0 1 1 1 0 0 1 1 0
```

N4 Accept

```
1
0
1 0
0 1
0 0
1 0 0 1 0 0
0 0 1 1 1 1 1 1 1 0 1 0 0
0 1 0 0 1 0 0 1 0 0
1 1 1 0 0
0 0 1 0
```

N4 Reject

```
0 0 0
1 1 0 0 0
0 0 0 1
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 1 1
1 1 0 1 0 1 0 0 0 0 0 0 1 0 1 1 1
1 0 1 0 0 1 0 0 0 1
0 0 0 0
0 0 0 0 0
```

N5 Accept

```
1 1
0 0
1 0 0 1
0 1 0 1
1 0 1 0
1 0 0 0 1 1 1 1 0 1
1 0 0 1 1 0 0 0 0 1 1 1 1 0 1 0
1 1 1 1 1
0 0 0 0
```

N5 Reject

```
0
1 1 1
0 1 0
0 0 0 0 0 0 0 0 0 0
1 0 0 0
0 1
1 0
1 1 1 0 0 1 0 1 0 0
0 1 0 1 1 1 1 1 1 1 1 1 0
0 0 0 1
0 1 1
```

N6 Accept

```
1 0
0 1
1 1 0 0
1 0 1 0 1 0
1 1 1
0 0 0 0 0 0
1 0 1 1 1
0 1 1 1 1 0 1 1 1 1
1 0 0 1 0 0 1 0 0
```

N6 Reject

```
1
0
1 1
0 0
1 0 1
0 1 1
1 1 0 0 1
1 1 1 1
0 0 0 0 0 0 0 0 0
0 1 0 1 1 1
1 0 1 1 1 1 0 1 1 1 1 1
1 0 0 1 0 0 1 0 0 1
```

N7 Accept

```
1
0
1 0
0 1
1 1 1 1 1
0 0 0
0 0 1 1 0 0 1 1
0 1 0 1
0 0 0 0 1 0 0 0 0 1 1 1 1
0 0 1 0 0
0 1 1 1 1 1 0 1 1 1 1 1
0 0
```

N7 Reject

```
1 0 1 0
0 0 1 1 0 0 1 1 0 0 0 0
0 1 0 1 0 1 0 1 0 1
1 0 1 1 0 1 0
1 0 1 0 1
0 1 0 1 0 0
1 0 1 0 0 1
1 0 0 1 0 0 1 1 0 1 0 1
```

8 References

Barnsley, M.F. 1988. *Fractals Everywhere*, (Academic Press, San Diego, CA).

Casey, M. 1996. The Dynamics of Discrete-Time Computation, with Application to Recurrent Neural Networks and Finite State Machine Extraction, *Neural Computation* **8**(6).

Casey, M. 1993. Computation Dynamics in Discrete-Time Recurrent Neural Networks, *Proceedings of the Annual Research Symposium of UCSD Institute for Neural Computation*, 78–95.

Cleeremans, A., D. Servan-Schreiber & J. McClelland, 1989. Finite State Automata and Simple Recurrent Networks, *Neural Computation*, **1**(3), 372–381.

Crutchfield, J.P. 1994. The Calculi of Emergence: Computation, Dynamics and Induction, *Physica D*, **75**, 11–54.

Crutchfield, J.P. & K. Young, 1990. Computation at the Onset of Chaos. In Zurek, W.H., ed. *Complexity, Entropy and the Physics of Information* (Addison-Wesley, Reading, MA).

Das, S. & M.C. Mozer, 1994. A Unified Gradient-Descent/Clustering Architecture for Finite State Machine Induction, *Neural Information Processing Systems* **6**, 19–26.

Fahlman, S.E. 1989. Fast-learning variations on back-propagation: an empirical study. In D. Touretzky, G. Hinton & T. Sejnowski, eds. *Proceedings of the 1988 Connectionist Models Summer School*, Pittsburgh, PA, 38–51 (Morgan Kaufman, San Mateo).

Forcada, M.L. & R.C. Carrasco, 1995. Learning the initial state of a second-order recurrent neural network during regular-language inference, *Neural Computation*, **7**(5), 923–930.

Frasconi, P., M. Gori & G. Soda, 1995. Recurrent Neural Networks and Prior Knowledge for Sequence Processing: A Constrained Nondeterministic Approach, *Knowledge Based Systems*, **8**(6), 313–332.

Giles, C.L., C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun & Y.C. Lee, 1992. Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks, *Neural Computation* **4**(3), 393–405.

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

Jordan, M.I. 1986. Attractor dynamics and parallelism in a connectionist sequential machine, *Proceedings of the Eighth Conference of the Cognitive Science Society*, Amherst, MA, 531–546.

Kolen, J.F. 1993. Fool's Gold: Extracting Finite State Machines from Recurrent Network Dynamics, *Neural Information Processing Systems* **6**, 501–508.

Kolen, J.F. 1994. Exploring the Computational Capabilities of Recurrent Neural Networks, Ph.D. Thesis, Ohio State University.

Lang, K.J. 1992. Random DFA's can be Approximately Learned from Sparse Uniform Examples, *Proc. Fifth ACM Workshop on Computational Learning Theory*, 45.

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

Omlin, C.W. & C.L. Giles, 1996. Extraction of Rules from Discrete-Time Recurrent Neural Networks, *Neural Networks*, **9**(1), 41.

Pollack, J.B. 1991. The Induction of Dynamical Recognizers, *Machine Learning* **7**, 227–252.

Pollack, J.B. 1987. Cascaded back propagation on dynamic connectionist networks, *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA, 391–404.

Rumelhart, D.E., G.E. Hinton & R.J. Williams, 1986. Learning representations by back-propagating errors, *Nature* **323**, 533–536.

Siegelmann, H.T. & E.D. Sontag, 1992. On the computational power of neural networks, *Proceedings of the Fifth ACM Workshop on Computational Learning Theory*, Pittsburgh, PA.

Tiňo, P., Bill G. Horne, C.L. Giles & P.C. Collingwood, 1995. Finite State Machines and Recurrent Neural Networks - Automata and Dynamical Systems Approaches, Tech. Rept. UMIACS-TR-95-1, Institute for Advanced Computer Studies, University of Maryland.

Tiňo, P. & J. Sajda, 1995. Learning and Extracting Initial Mealy Automata with a Modular Neural Network Model, *Neural Computation* **7**(4), 822.

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

Trakhtenbrot, B.A. & Ya.M. Barzdin' 1973. *Finite Automata; Behavior and Synthesis* (North-Holland, Amsterdam).

Watrous, R.L. & G.M. Kuhn, 1992. Induction of Finite State Languages Using Second-Order Recurrent Networks, *Neural Computation* **4**(3), 406–414.

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

Zeng, Z., R.M. Goodman & P. Smyth, 1994. Learning Finite State Machines with Self-Clustering Recurrent Networks, *Neural Computation* **5**(6), 976–990.