Communicated by Raymond Watrous

First-Order Recurrent Neural Networks and Deterministic Finite State Automata

Peter Manolios*
Department of Computer Science, Brooklyn College, Brooklyn, NY 11210 USA and Department of Computer Science, CUNY Graduate Center and University Place, New York, NY 10036 USA

Robert Fanelli
Department of Physics, Brooklyn College, Brooklyn, NY 11210 USA

We examine the correspondence between first-order recurrent neural networks and deterministic finite state automata. We begin with the problem of inducing deterministic finite state automata from finite training sets, that include both positive and negative examples, an NP-hard problem (Angluin and Smith 1983). We use a neural network architecture with two recurrent layers, which we argue can approximate any discrete-time, time-invariant dynamic system, with computation of the full gradient during learning. The networks are trained to classify strings as belonging or not belonging to the grammar. The training sets used contain only short strings, and the sets are constructed in a way that does not require a priori knowledge of the grammar. After training, the networks are tested using various test sets with strings of length up to 1000, and are often able to correctly classify all the test strings. These results are comparable to those obtained with second-order networks (Giles *et al.* 1992; Watrous and Kuhn 1992a; Zeng *et al.* 1993). We observe that the networks emulate finite state automata, confirming the results of other authors, and we use a vector quantization algorithm to extract deterministic finite state automata after training and during testing of the networks, obtaining a table listing the start state, accept states, reject states, all transitions from the states, as well as some useful statistics. We examine the correspondence between finite state automata and neural networks in detail, showing two major stages in the learning process. To this end, we use a graphics module, which graphically depicts the states of the network during the learning and testing phases. We examine the networks' performance when tested on strings much longer than those in the training set, noting a measure based on clustering that is correlated to the stability of the networks. Finally, we observe that with sufficiently long training

*Current address: Department of Computer Sciences, Taylor Hall, University of Texas at Austin, Austin, TX 78712-1188 USA.

times, neural networks can become true finite state automata, due to the attractor structure of their dynamics.

1 Introduction

The NP-hard problem of inferring deterministic finite state automata (DFA) from finite training sets that include both positive and negative examples has been addressed by many researchers. For an up-to-date listing of algorithmic inference methods see Miclet (1990). Researchers have also trained both first- and second-order recurrent neural networks to induce grammars. When speaking of "grammars," we will refer only to regular grammars, which are equivalent to finite state automata. For a proof of this equivalence, and for an introduction to formal language theory, see Harrison (1978). We use recurrent networks to infer DFA because recurrent networks have a memory and can process strings of any length. For a more detailed discussion on the advantages of recurrent networks, see Elman (1990). Furthermore, we argue that the recurrent architecture we use is a universal approximator for discrete-time, time-invariant dynamic systems, and can therefore approximate any DFA.

The researchers who have used first-order recurrent networks, have concentrated on the Reber (1967) grammar, and have trained networks to predict the next symbol of a string generated from the grammar. Cleeremans *et al.* (1989) used a simple recurrent network architecture proposed by Elman (1990), computing a truncated gradient during learning, and succeeded in training networks to learn the Reber grammar. They had some difficulty with the embedded Reber grammar, for which Smith and Zipser (1989), who used real time recurrent learning that computes the full gradient, were able to successfully train networks.

The researchers who used second-order recurrent networks, concentrated on the Tomita (1982) grammars, and instead of training networks to predict the next symbol of a string, they trained networks to decide whether a string is grammatical. Watrous and Kuhn (1992a), using a learning algorithm that computes the full gradient, were able to train networks that can generalize for a subset of the Tomita grammars. Giles *et al.* (1992), using a real time forward training algorithm that also computes the full gradient, had successful results for Tomita grammar 4, and reported that similar results were obtained for the other Tomita grammars. Giles *et al.* (1992) and Omlin *et al.* (1992) also presented a finite state automaton extraction algorithm used to extract the network's conception of the finite state automaton it is learning. Fanelli (1993) has trained Elman type nets to infer some Tomita grammars using full gradient computation.

We consider the Tomita grammars over $\{0,1\}^*$, since work in this area has tended to focus on these grammars. Specifically, we consider the

following subset of the seven Tomita (1982) grammars:

L1. 1*

L2. (10)*

L4. no more than two 0s in a row

L6. number of 1s − number of 0s $\equiv 0 \pmod 3$

The architecture, learning, and training procedures are discussed in the next two sections. A finite state automaton extraction algorithm similar to that of Giles *et al.* (1992) is presented in Section 4. In order to better understand the behavior of the networks, we develop a graphics algorithm (Section 5) that allows us to depict various aspects of the networks' behavior. In the results section, we show that the architecture is capable of learning the Tomita grammars studied, with only a restricted number of examples. Also in the results section, we analyze the learning process in detail, examining clustering during and after training and testing, noting a feature of the networks based on clustering, which is correlated to their stability, and giving an example of a neural network that becomes a DFA.

2 Architecture

Motivated by Hornik *et al.* (1989), who proved that feedforward networks with one hidden layer are universal approximators, we argue that the architecture we consider is the simplest first-order architecture with two recurrent layers that is a universal approximator for discrete-time, time-invariant dynamic systems. Such a system can be described by the following formulas:

$$\begin{aligned} \mathbf{y}(t) &= \Omega[\mathbf{x}(t-1), \mathbf{u}(t-1)] \\ \mathbf{x}(t) &= \Phi[\mathbf{x}(t-1), \mathbf{u}(t-1)] \end{aligned}$$

where $\mathbf{y}(t) \in \Re^n$ is the output of the system at time t, $\mathbf{x}(t) \in \Re^m$ is the state of the system at time t, $\mathbf{u}(t) \in \Re^r$ is the input of the system at time t, and t is a nonnegative integer. With two feedforward networks, one which approximates Ω and the other Φ, we can approximate any discrete-time, time-invariant dynamic system. The resulting architecture is shown in Figure 1a and b. If we connect Y to H2, H2 to X, Y to H1, and H1 to Y, the resulting architecture (shown in Fig. 1c) loses no generality because with weights of 0 for all the added connections, it is the same as before, i.e., Figure 1b is a specific case of Figure 1c. Note that layers H1 and H2 of Figure 1c have the same connections and can be combined into one hidden layer, and similarly, layers X and Y can be combined into a single state-output layer. The resulting three layer recurrent network architecture, shown in Figure 1d and e, consists of input, hidden, state,

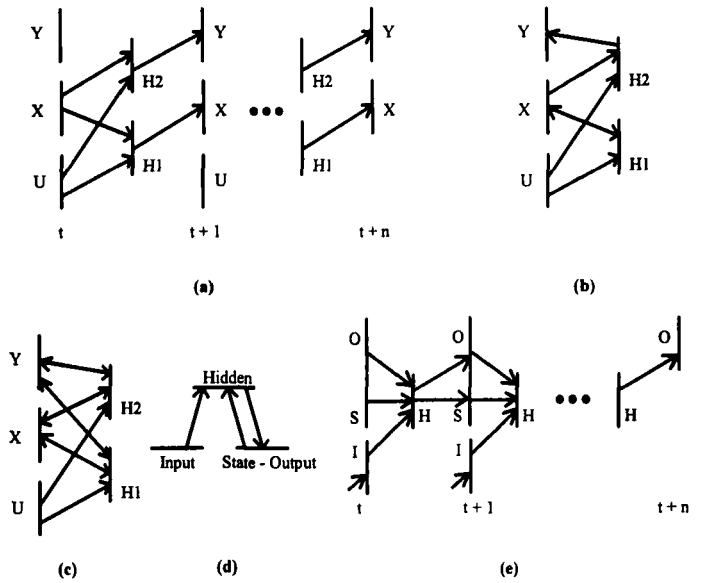


Figure 1: (a) and (b) are two views of an architecture that is a universal approximator for the two functions Ω and Φ that describe a discrete-time, time-invariant dynamic system. By adding connections from Y to H2, H2 to X, Y to H1, and H1 to Y, we obtain the structure (c), which reduces to the architecture of (d) and (e). In (e), the three layers are spread out and shown during successive time steps $t, t+1, \ldots, t+n$. The input units, together with the state units, feed into the hidden units. The hidden units in turn modify the state units. A nonempty subset of the state units is called the output unit. These are treated exactly as the state units, except that their activations are considered the output of the network. A bias unit connected to all the other units is also part of the architecture, but is not shown.

and bias units, and is the architecturally simplest first-order network with two recurrent layers that is a universal approximator for discrete-time, time-invariant dynamic systems. A subset of the state units corresponds to the output units, which are treated exactly as the state units, except that their activations are considered the output of the network. The activations of units can be any value in the closed interval $[0, 1]$.

3 Learning Algorithm and Training

We train the architecture with full gradient computation, using backpropagation through time (Rumelhart *et al.* 1986). An epoch consists of the

presentation of all strings in the training set once. Batch updating is used, i.e. the error is accumulated and the weights of the network modified at the end of each training epoch so that the order in which the training set is presented is immaterial.

The architecture is trained on a subset of the Tomita grammars, specifically grammars one, two, four, and six. No attempt is made to train the architecture on the other Tomita grammars. One input unit, two state units with one designated as output, and two hidden units are used for Tomita 1 and 2, while three hidden units are used for Tomita 4 and 6. Therefore, for Tomita 1 and 2 there are 14 trainable weights, and for Tomita 4 and 6, there are 20, including the biases. The learning rates and momentum terms used for Tomita grammars 1, 2, and 4 are held constant during training at 0.1 and 0.5, respectively. The learning rates and momentum terms used for Tomita grammar 6 are 0.1 and 0.5, respectively, for the first 1K epochs, and are then set to 0.01 and 0.05, respectively, for the rest of training. For this application, an activation of 1 at the output unit is used to indicate a grammatically correct string, and an activation of 0 a grammatically incorrect string.

The set of training strings used is small and their length short, so that we can establish a tight upper bound on the information required by a network in order to achieve good generalization ability. The training sets used for all the Tomita grammars consist only of all strings up to length four, including the NULL (empty) string. This means that the training sets of Tomita 1 and Tomita 2 contain only five and three grammatically correct strings, respectively. In spite of this, our networks are able to not only correctly classify all the strings in the training set, but are also able to correctly classify strings of arbitrary length, as discussed in Section 6. In addition, the networks can induce the minimal DFA that correspond to the grammars. The networks had some difficulty with Tomita 6 because there are many local minima. To avoid the local minima, we train networks for 1K epochs and select for further training the first four networks whose RMS error is less than 0.16. Other work in the field has tended to use much longer training strings, and more elaborate procedures for modifying the frequency with which strings in the training set are presented. Both of these procedures help keep the network out of local minima, thereby making it easier for the network to converge, but in our case we find that such procedures are not necessary.

4 Deterministic Finite State Automaton Extraction Algorithm

Motivated by Cleeremans *et al.* (1989) who used cluster analysis to determine the internal representations of their networks, and Giles *et al.* (1992) and Watrous and Kuhn (1992b) who present processes for extracting FSA from second-order networks, we have developed a vector quantization based DFA extraction algorithm similar to that of Giles *et al.* (1992), Wa-

trous and Kuhn (1992b), and Zeng *et al.* (1993). This algorithm is used to determine the DFA that the network is approximating. The networks we obtain are usually able to induce the minimal DFA for all the Tomita grammars we consider. Some DFA induced are not minimal, but are reducible to the minimal DFA using a standard DFA minimization algorithm (Harrison 1978).

Following the spirit of the discussion of the architecture, we use the state units alone to represent the state of the system being approximated. If the cardinality of the set of state units is β, and the activations of units can be any value in the closed interval $[0, 1]$, then the activations of the state units can be mapped into a point in a β-dimensional unit hypercube. We will call such points network states, and we will call the hypercube the state space. When all network states corresponding to the activations of the set of state units in response to a set of test strings are plotted, it is observed that the network states are not uniformly distributed, and in fact tend to cluster together. The DFA extraction algorithm can confirm that these clusters correspond to the states of a DFA that is being approximated.

The DFA extraction algorithm is given as input the sequence of network states visited by the network in the β-dimensional hypercube, in response to a test set, and analyzes the network states, trying to identify the clusters. This is done by randomly distributing n markers within the hypercube. For every network state, the closest marker is moved toward the network state a certain distance d, which is equal to the distance between the marker and the network state divided by the number of times the marker has been moved plus one. This guarantees that any marker moved will be exactly in the centroid of the network states it is closest to. Now, every marker moved should represent one cluster of network states. The DFA extraction algorithm also verifies that the clusters found can be identified with states in a DFA, i.e., as a network state is associated with a certain cluster, the algorithm checks that the possible transitions out of the network state are exactly the same as those out of the other network states in the cluster. For example, if one marker represents two clusters, then the network states of one cluster will have different transitions than the network states of the other cluster, and it will seem that we have a nondeterministic finite state automaton. The algorithm will reject this result and will restart itself with a different set of randomly distributed markers. Several such consistency checks are performed, and statistics are compiled giving the centroid, standard deviation, and distance to the farthest point in each cluster. If a deterministic automaton can be extracted, then a state table that identifies the start state, all accept states, all transitions into and out of states, as well as all the above mentioned statistics is printed.

Note in general it is not possible to explore the state space completely since there are a countably infinite number of possible test strings in $\{0,1\}^*$, and different test sets will cause the network to visit different sets

of network states. Hence, it is possible that given two different test sets, the algorithm will succeed in extracting a DFA for one of the sets, and fail for the other. We have obtained such results with networks that have limited generalization ability. Thus, it is useful that a DFA extraction algorithm allow for alternate explorations of the state space, and this is an important feature of our algorithm.

5 Graphics

In order to further understand the behavior of the networks, we have developed a graphics algorithm that we use to view the behavior of the networks during training and testing. The graphics algorithm displays only a two-dimensional view of the activation space of the state units (hypercube), but since the state space of our networks is only two-dimensional, this does not pose any difficulties.

To view the behavior of the networks during training, the algorithm requires several input sequences that depend on the training set. The first sequence is provided by the DFA extraction algorithm, which partitions all the network states in the state space of the fully trained network in response to the training set, into clusters, and outputs a sequence of pairs that consist of network states and the clusters they belong to. The remaining sequences consist of the coordinates of network states resulting from the presentation of the training set to the network at varying degrees of training. The first input sequence is used to plot all network states belonging to the same final cluster with the same color and to plot each cluster with a different color. The other sequences are used to show the evolution of states visited by the network as training progresses. That is, initially the state space of the network without training is presented, then the state space after a few epochs, and so on until at last the state space of the network after all learning has been completed is presented. The result is an animated graphic presentation of the learning process.

Using the graphics algorithm to view networks during training, we observe that the networks go through two stages, which we call the decide and reinforce stages. We will have more to say about the two stages in the next section so we will only give a quick overview now. During the decide stage, the network decides which DFA it will approximate, and during the reinforce stage, the network strengthens its approximation of that DFA. The reinforce stage consists of tightening the clusters, or equivalently forcing every network state in a cluster toward the cluster centroid.

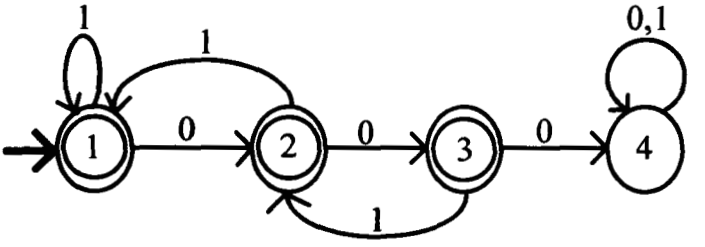
To view the behavior of the networks during testing, the algorithm requires input sequences that depend on the training set and several test sets. All sequences are provided by the DFA extraction algorithm, with the requirement that for all the test sets, the extracted DFA are isomorphic, a requirement met by all the data considered. The first sequence

is the same as the first sequence discussed above. Each of the other sequences consists of the coordinates of network states resulting from the presentation of one of the test sets to the fully trained network. As before, the first input sequence is used to plot all network states belonging to the same cluster with the same color and to plot each cluster with a unique color. The remaining sequences are used to show the network states in the hypercube visited by the network as longer strings are considered. That is, initially the network states visited while processing the training set are presented, then the network states visited as some new strings are considered, and so on until at last the state space of the network after all test strings have been considered is presented. Using the graphics algorithm with test sets consisting of many more strings than the training set allows us to graphically view the performance of the network on strings that it has never before encountered. Our training sets consist of all 31 strings of length ≤ 4, and the test sets we use with the graphics algorithm consist of all strings of length $\leq 5, \leq 6, \ldots, \leq 12$. The result is an animated graphic presentation of the testing process.

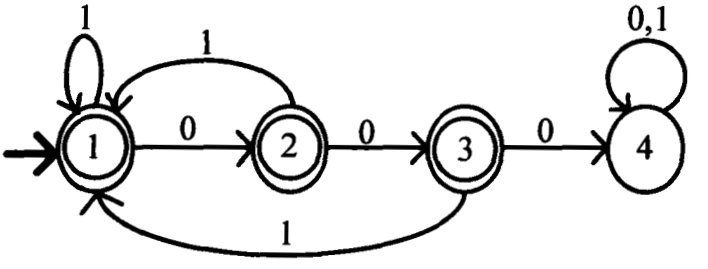
6 Results

For all the Tomita grammars we consider, we train randomly initialized networks on the training set (all strings of length ≤ 4), test the networks' generalization by using various test sets, and extract DFA for each test set using the DFA extraction algorithm. For Tomita grammars 1, 2, and 4, we use eight test sets, namely all strings of length $\leq 5, \leq 6, \ldots, \leq 12$. Many of our networks can correctly classify all strings in the test sets and infer a correct DFA, which retains its structure for all the test sets. For Tomita 6, we use the eight previously mentioned test sets as well as two additional test sets: 1000 random strings of length ≤ 100, and 1000 random strings of length ≤ 1000. Our networks have the most difficulty learning Tomita 6 because there are many local minima. Given the difficulty of Tomita 6, our results will focus on this grammar.

One consequence of using a training set as small as we do, is the possibility that there is more than one minimal DFA that accepts all the positive and rejects all the negative examples. This is the case with Tomita 4, where from two different networks trained on the same training set, we extract two different minimal DFA (Fig. 2). This is an important point because if a network is not given enough examples to constrain the grammar in question, then the network may not be able to generalize. One would then conclude that the network is unable to learn the grammar, but this is not a valid conclusion, since with a different training set it is possible that the network will learn the grammar. Watrous and Kuhn (1992a) suggest that the reason why some of their networks are not able to generalize is that the grammar is not sufficiently constrained by their training sets. However, knowing if a training set contains enough



(a)



(b)

Figure 2: The two DFA shown (a,b) were obtained from two different networks trained on Tomita 4 with the same training set, using the DFA extraction algorithm described previously. The heavy arrow indicates the start state, and accept states are indicated by two circles. Notice that both will correctly classify all strings of length ≤ 4, but (a) will not be able to correctly classify the string 00100.

strings to uniquely constrain the set of possible minimal grammars requires some knowledge of the grammar in question, and in many cases such knowledge is not available. See Fanelli (1993) for further discussion.

Tomita 6 is the hardest grammar for our networks to learn, but even so, network 1 correctly classifies all strings in the test sets to within 0.02 of the target output. Note that the networks are trained using only the training set and that after all learning is over, the test sets are used to

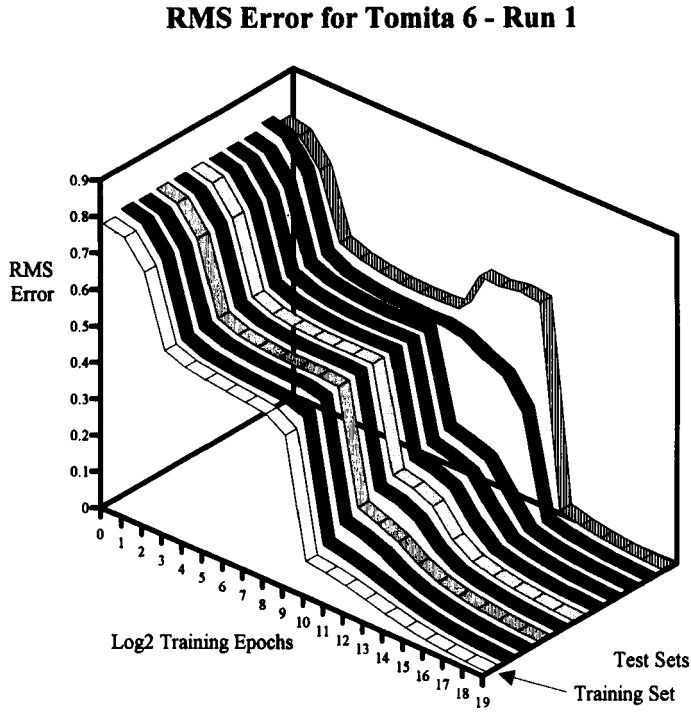


Figure 3: This graph shows the RMS errors for network 1 on the training set and on various test sets. The network is trained for 2^{19} epochs on all strings whose length is ≤ 4. The state of the network at various points in training is saved, and later tested on ten test sets. The test sets are all strings of length $\leq 5, \leq 6, \ldots, \leq 12$, 1000 random strings of length ≤ 100, and 1000 random strings of length ≤ 1000. The results for the training set and each test set are shown from left to right, respectively. The fully trained network is able to correctly classify all the strings in all the test sets, with a maximum RMS error of 0.0075.

gauge the generalization ability of the networks. Figure 3 shows the root mean squared error for network 1.

The DFA extracted from network 1 is shown in Figure 4 along with the minimal DFA for Tomita 6, to which it can be reduced by standard methods. Each of the six states of the extracted DFA corresponds to a cluster, a set of network states in the β-dimensional hypercube. Recall that the DFA extraction algorithm determines the centroids and standard deviations of these clusters. We introduce the term "tightness" of a clus-

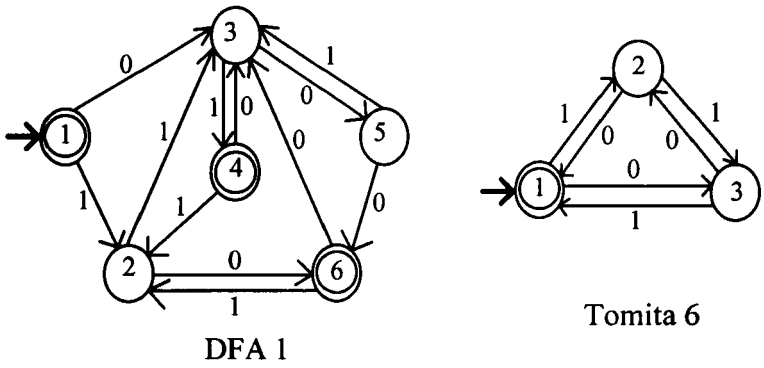


Figure 4: DFA 1 is the actual DFA extracted from network 1, using the DFA extraction algorithm. Using a standard minimization algorithm, we can reduce it to the minimal DFA for Tomita 6.

ter, which is inversely related to the standard deviation of the geometric distances of the network states from their centroid. The smaller this measure, the "tighter" the cluster, and the larger this measure, the "looser" the cluster.

Using the graphics algorithm of the previous section to view network 1 (Figs. 3 and 4) during training, we see (Fig. 5) that initially all network states are in one cluster, but as training continues, the network states start to move around, even traveling from one end of the hypercube to the other as they form clusters. After 2^{10} training epochs, the clusters stay fixed and start to tighten, i.e., converge to the cluster centroid. Note that cluster 1 of network 1 contains only the fixed network start state and can therefore never change.

All our networks exhibit similar behavior, and to analyze these results, we use the following procedure. We extract the DFA from the trained network, noting which network states belong to which clusters. We then look at the network during various points in the training process, grouping network states that will be in the same cluster into sets, and calculate the standard deviations of the resulting sets at these points. The results of this procedure on network 1 are presented in Figure 6, where we notice that the standard deviations of the clusters behave erratically at first, and then start to approach zero.

From these results, we conclude that there are two stages during learning: the decide and reinforce stages. We observe that during the decide stage, the cluster centroids move around in the space. Once the network has decided on the DFA it will approximate, there is a transition to the re-

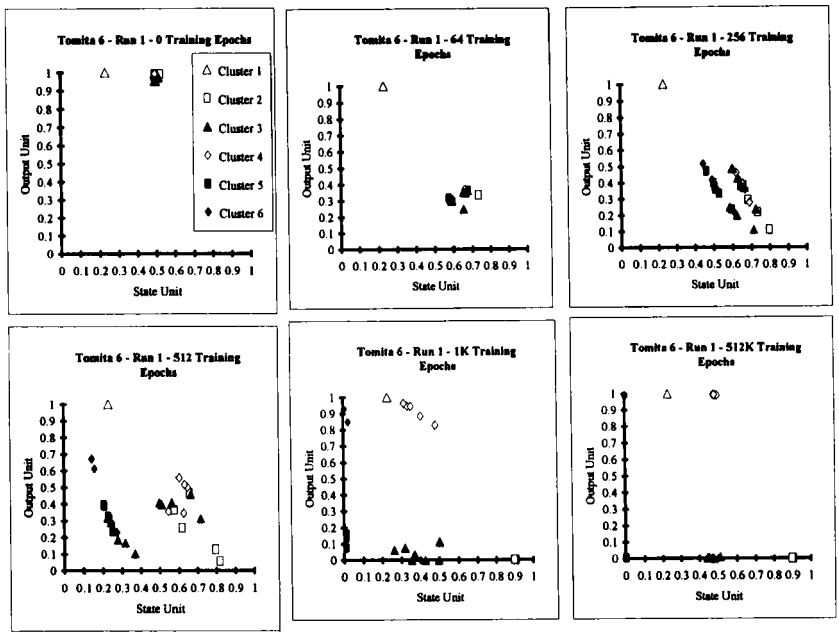


Figure 5: This figure shows the network states of network 1 during training, showing the development of clusters that correspond to network states. We find that initially all points cluster together, and that eventually they spread apart, so that both network states and clusters travel through the state space until they settle on a certain location. This is part of the decide stage and in this case lasts about 2^{10} (1K) epochs, as can be seen in the diagram.

inforce stage, where it strengthens its approximation of the DFA. During the reinforce stage, individual network states converge to the cluster centroid, hence the standard deviation approaches zero. This transition can be observed when the standard deviations of the clusters stop behaving erratically and start to get smaller (at around 2^{10} epochs in Fig. 6).

Having seen what happens during training, we consider what happens after the training is over, when networks are tested on strings they have not previously encountered. For this phase of analysis, we test the networks first on the training set, and then on all strings up to length 5, length 6, ..., length 12, and for Tomita 6, 1000 random strings of length ≤ 100, and 1000 random strings of length ≤ 1000. Using the DFA extraction algorithm, we find that networks that learn the training set are approximating DFA and that many of these DFA remain stable for all the test sets. The graphics algorithm is used to confirm this (see Fig. 7 for the

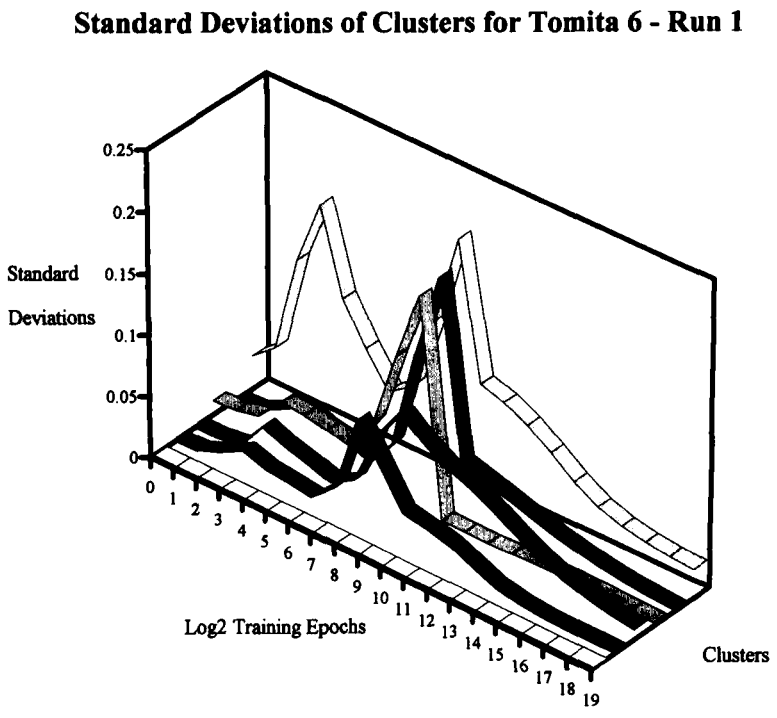


Figure 6: This graph shows the standard deviations of the six clusters of network 1 (Figs. 3, 4, and 5) during training. From left to right we have clusters 1, 5, 4, 2, 6, and 3. Large changes in the standard deviations up to 2^{10} epochs reflect the movement of network states and clusters during the decide stage. The 2^{10} epoch marks the transition from the decide to the reinforce stage, within which the network strengthens the approximation of the DFA it has decided on. During the reinforce stage, the clusters stay fixed in the state space and the graph shows the network states in the clusters converging.

example of network 1). We find that for network 1, the DFA structure not only remains stable, but the clusters do not grow in size, even when tested on strings with up to 1000 characters. To confirm this performance, we test network 1 using two different simulators written independently by each of the authors. The simulators run on different computer architectures and their results for a test set containing 1000 random strings of length ≤ 100 are identical. Analysis of the networks' behavior on test sets that include very long strings suggests that as clusters are revisited, new network states are generated, but at a certain point, the clusters become

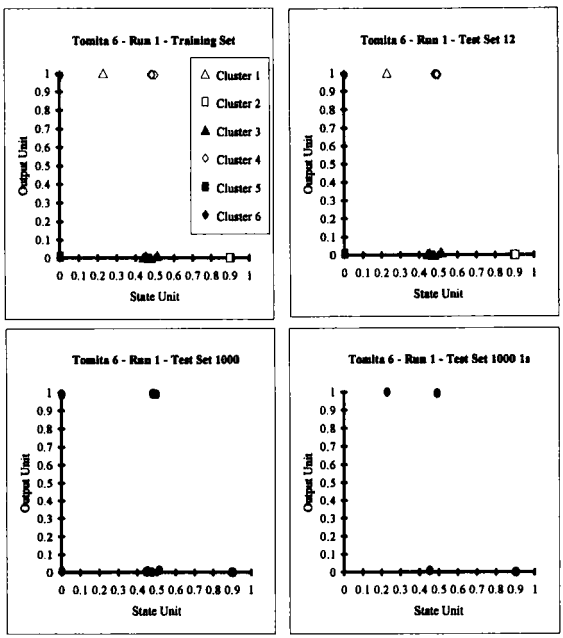


Figure 7: These graphs show how network 1 behaves during testing. Test Set 12 contains all strings of length ≤ 12. Test Set 1000 contains 1000 random strings of up to length 1000. Note that Test Set 1000 does not contain cluster 1 because cluster 1 contains only the fixed network start state and the empty string is not in the test set. For similar reasons, Test Set 1000 1s does not generate clusters 5 or 6. The DFA extraction algorithm is used to partition the network states resulting from the presentation of the Training Set and Test Set 12.

absolutely stable, neither growing nor moving, perhaps shrinking. At that point, the network is in effect a DFA. We suspect that some of this is due to numerical round-offs inherently associated with limited computer precision, but that the primary effect is due to the attractor structure of the network considered as a dynamic system. An example of this behavior can be seen in Figure 7. When the network is presented with a string of 1000 1s, after the twenty-third 1, the same three points are revisited in succession, within the precision of the floating point processor. Thus we conclude the simulation has entered a limit cycle, but due to machine round-off, we cannot say exactly what kind of attractor the ideal network would have entered.

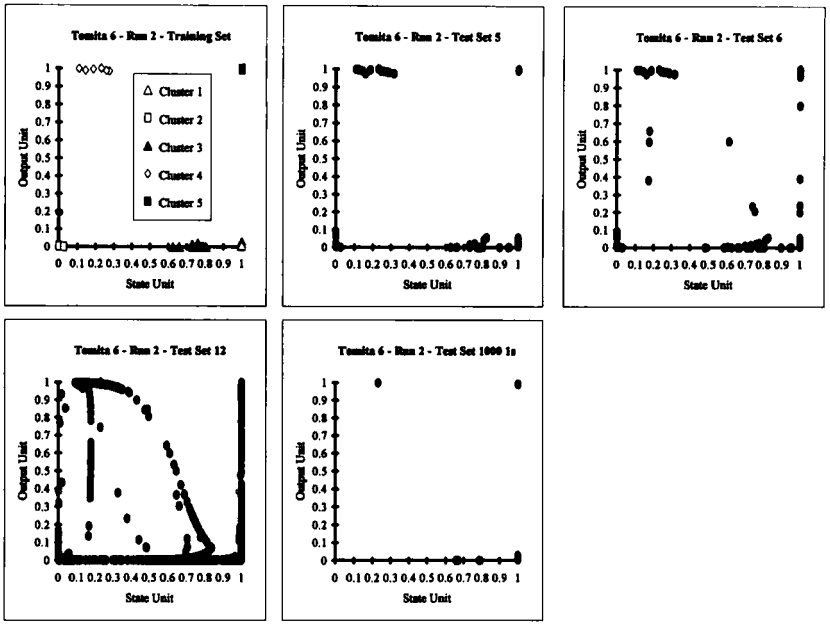


Figure 8: These graphs show how network 2, which performs flawlessly on the training set but is not able to generalize well, behaves during testing. Using the DFA extraction algorithm, we can extract the clusters shown for the Training Set (all strings of length ≤ 4). However, the DFA extraction algorithm fails to find a consistent DFA for Test Set 5 (all strings of length ≤ 5) because clusters 1 and 3 have loosened. For Test Sets 6 and 12 (all strings of length ≤ 6, ≤ 12), there are no identifiable clusters, but as Test Set 1000 1s (all strings in the grammar 1* of length ≤ 1000) shows, the network can retain its automaton structure for a subset of the grammar.

All four networks trained on Tomita 6 can correctly classify all strings in the training set, however, only two become DFA. An example of a network that does not become a DFA is network 2, for which we can extract a DFA only for the training set. When we view the behavior of this network on various test sets, we see that the network's clusters quickly lose their cohesion (Fig. 8). However, when tested on a string of 1000 1s, we see that network 2's performance is flawless. Recall that the network is emulating the two functions Ω and Φ, which map the current state and input to the next output and state, respectively. Any particular DFA cor-

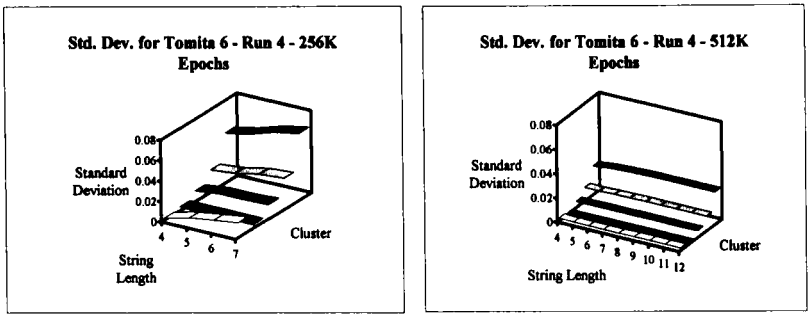


Figure 9: The two graphs show the standard deviations of the clusters for network 4, trained on Tomita 6 for 2^{18} (256K) and 2^{19} (512K) epochs, when tested on all strings of length $\leq 4, \ldots, \leq 12$. Both versions of the network induced equivalent DFA, and corresponding clusters are shown in the same order. These graphs illustrate that there is a threshold and when the standard deviations of network clusters are below the threshold, as is the case with the clusters of network 4 trained for 2^{19} (512K) epochs, they do not grow further. The standard deviations of the clusters of network 4, trained for 2^{18} (256K) epochs, are not below the threshold, and therefore expand to the point where the DFA extraction algorithm cannot extract a DFA for test sets containing all strings of length 8 and above.

responds to a class of such functions that emulate it either approximately or, through their attractor structure, exactly. What Figure 8 shows is that network 2 approximates these two functions exactly for only a proper subset of their domains.

We observe that as training is increased, network clusters tighten and converge to the cluster centroid. Figure 9 shows the standard deviations of the clusters for network 4 when fully trained and when trained for only 2^{18} epochs. When clusters are sufficiently tight, they do not grow further, i.e., there is a threshold and when cluster standard deviations are below the threshold, then the clusters do not loosen. On the other hand, when cluster deviations are above the threshold, the clusters loosen as the network is presented with longer strings. As Figure 9 shows, the clusters of network 4 trained for 2^{19} epochs are tight, and the network is able to retain its automaton structure (Fig. 10). However, the clusters of network 4 trained for 2^{18} epochs are loose (Fig. 9); they start to overlap with other clusters, and the network's state structure becomes poorly defined (Fig. 10).

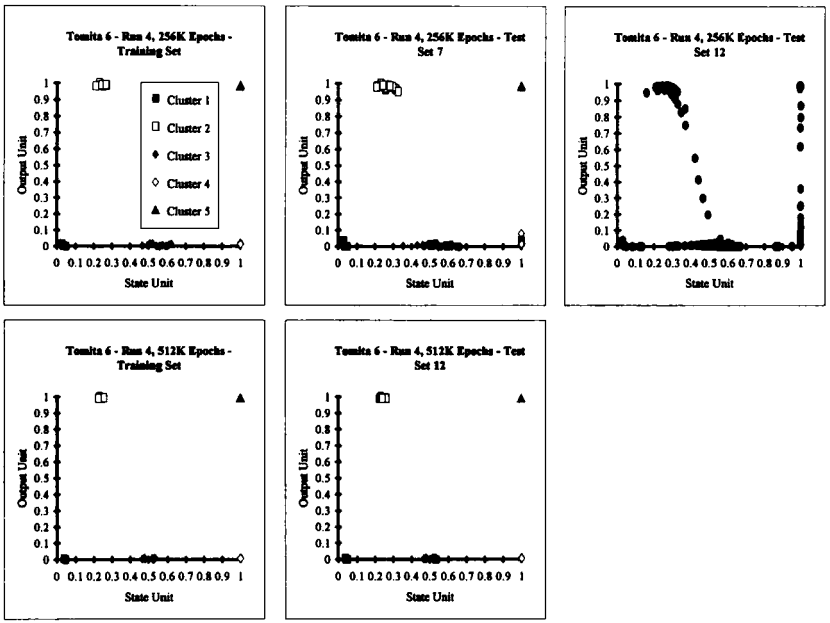


Figure 10: These graphs show how two versions of network 4, one trained for 2^{18} (256K) epochs, and the other for 2^{19} (512K) epochs behave when tested on various test sets. The clusters of the 256K network are not tight enough to be below the threshold, and the network is not able to retain its automaton structure. After additional training, the standard deviations of the resulting 512K network's clusters are below the threshold, and the network is able to retain its automaton structure indefinitely. The Training Set contains all strings of length ≤ 4, and Test Sets 7 and 12 contain all strings of length ≤ 7 and ≤ 12, respectively.

7 Discussion

Given that our architecture is arguably a universal approximator for discrete-time, time-invariant dynamic systems, we expect, in principle, that it can learn the Tomita grammars, and we see in practice, that using very small networks containing only a few units, it has been able to learn all of the Tomita grammars attempted. The size of the training sets has been kept small (31 strings), and the training strings used have been short (length ≤ 4), in comparison to those used in other studies. Many networks can generalize from the training sets to strings of length

12. Some can become DFA and therefore remain stable and generalize perfectly on strings of any length.

We use the DFA extraction algorithm, which has been implemented as a C program, to determine whether the networks are inducing deterministic finite state automata. The algorithm gives an excellent indication of how well the network approximates a particular DFA when processing a given test set of strings, but as we point out in Figures 9 and 10, two networks that have induced equivalent DFA on the training set can generalize to varying degrees. By comparing the standard deviations of clusters, our DFA extraction algorithm allows us to distinguish between such networks. Note that once a DFA is extracted, one can write a program to simulate the DFA, resulting in the obvious advantages that the program will always be correct, and will also be more efficient.

We use the DFA algorithm in combination with the graphics algorithm to view the development and modification of the clusters during training and testing of the networks. We observe two stages during learning: the decide and reinforce stages. During the decide stage, the network settles on a specific DFA. During the reinforce stage, the clusters shrink and the network strengthens its approximation of the DFA.

We use the standard deviation of the network states of a cluster as a quantitative measure of its tightness. The smaller this measure, the tighter the cluster. We have found that in certain networks, when the standard deviation of the clusters reaches a threshold, the network becomes a DFA and is capable of correctly classifying strings of arbitrary length. The networks are therefore capable of absolute stability and flawless generalization ability. The ability of some extensively trained (2^{19} epochs) networks to do this is a notable result of this work. We note that Hirsch (1989) has discussed the possibility of the correspondence of the attracting equilibria of convergent recurrent networks to the states of a finite automaton and of the induction of such a correspondence through training. The details and some further instances of this phenomenon are currently under investigation. We think that the above work helps to clarify the relationship between DFA and recurrent neural networks trained to emulate them.

References

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

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

Elman, J. L. 1990. Finding structure in time. *Cog. Sci.* **14**, 179–211.

Fanelli, R. 1993. *Grammatical Inference and Approximation of Finite Automata by Simple Recurrent Neural Networks Trained with Full Forward Error Propagation.* Tech. Rep. NNRG 930811A, Dept. of Physics, Brooklyn College of the City University of New York.

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

Harrison, M. H. 1978. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA.

Hirsch, M. W. 1989. Convergent activation dynamics in continuous time networks. *Neural Networks* **2**(5), 331–349.

Hornik, K., Stinchcombe, M., and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* **2**, 359–366.

Miclet, L. 1990. Grammatical inference. In *Syntactical and Structural Pattern Recognition; Theory and Applications*, H. Bunke and A. Sanfeliu, eds., Chap. 9. World Scientific, Singapore.

Omlin, C. W., Giles, C. L., and Miller, C. B. 1992. Heuristics for the extraction of rules from discrete-time recurrent neural networks. *Int. Joint Conf. Neural Networks* **I**, 33–38.

Reber, A. S. 1967. Implicit learning of artificial grammars. *J. Verbal Learning Verbal Behav.* **6**, 855–863.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. Learning internal representations by backpropagating errors. *Nature (London)* **323**, 533–536.

Smith, A. W., and Zipser, D. 1989. Encoding sequential structure: Experience with real-time recurrent learning algorithm. *Proc. Int. Joint Conf. Neural Networks* **I**, 645–648.

Tomita, M. 1982. Dynamic construction of finite automata from examples using hill-climbing. *Proc. Fourth Int. Cog. Sci. Conf.* pp. 105–108.

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

Watrous, R. L., and Kuhn, G. M. 1992b. Induction of finite-state automata using second-order recurrent networks. In *Advances in Neural Information Processing Systems 4*, J. Moody *et al.*, eds., pp. 309–316. Morgan Kaufmann, San Mateo, CA.

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

Zeng, Z., Goodman, R. M., and Smyth, P. 1993. Self-clustering recurrent networks. *IEEE Int. Conf. Neural Networks* **I**, 33–38.

Received September 7, 1993; accepted February 10, 1994.