RECURRENT NETWORKS: STATE MACHINES OR ITERATED FUNCTION SYSTEMS?

John F. Kolen
Laboratory for AI Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210
kolen-j@cis.ohio-state.edu

Introduction

Feedforward neural networks process information by performing fixed transformations from one representation space to another. Recurrent networks, on the other hand, process information quite differently. To understand recurrent networks one must confront the notion of state as recurrent networks perform iterated transformations on *state representations*. Many researchers have recognized this difference and have suggested parallels between recurrent networks and various automata[1, 2, 3]. First, I will demonstrate how the common notion of deterministic information processing does not necessarily hold for deterministic recurrent neural networks whose dynamics are sensitive to initial conditions. Second, I will link the mathematics of recurrent neural network models with that of iterated function systems [4]. This link points to model independent constraints on the recurrent network state dynamics that explain universal behaviors of recurrent networks like internal state clustering.

Recurrent Networks as State Machines

Current models of human intellectual competence demand the potential to store and act upon any one of an infinite collection of internal representations (e.g., [5, 6]). Inspired by the pioneering work of McCulloch and Pitts [7], several simulation proofs of universal computation in recurrent networks have appeared over the last few years [8, 9, 10, 11]. Each of these projects clearly supports the claim of representational adequacy for recurrent networks by showing that recursive computation can arise by *design*. These projects, however, provide little insight in the *detection* of computation occurring in a system. In other words, they don't tell us how to recognize computation when it is happening before our eyes.

The attribution of computational behavior to a system traditionally rests on the discovery of an isomorphism between states of the physical (or dynamical) system to the information processing states of the computational model [12]. Certain voltage levels of particular wires in a CPU, for instance, are isomorphic to the bits processed by the machine. A modeler must measure the current state of a process with sufficient resolution to justify the isomorphism. In models capable of recursive computation, the information processing state can demand unbounded accuracy from the modeler's measurements.

Many researchers studying the computational power of recurrent networks have followed this tradition. These researchers clustered internal state vectors and drew correspondences between these clusters and the deterministic state machine generating the training data [13]. This work was extended and can be summarized as variants of dynamic state partitioning [1]. While Giles and his collaborators use arbitrary partitions of state space to build clusters, others employ statistical clustering techniques to extract their state clusters[2]. Each of these methods shares the same problem of multiple inductions; a single recurrent network can produce many state machine interpretations.

The work of Crutchfield and Young [14] suggests that information processing states can be assigned without highly precise examination of the internal dynamics. Rather than assuming a stream of noisy numerical measurements, they construct models from periodic samplings with a single decision boundary. The binary sequence they collect requires a computational description (i.e. what kind of automaton could have generated the sequence), unlike numerical measurements that are often described mathematically. The minimal finite state automaton induced from this sequence of discrete measurements provides a realistic assessment of the intrinsic computational complexity of the physical system under observation.

An important distinction can be drawn between the methods of Crutchfield and Young with those used by the multitude of researchers studying recurrent networks. Dynamical systems theory, particularly the study of systems producing chaotic behavior, demonstrates that significant "state" information can be buried deep within the system's initial conditions. This sensitivity to initial conditions implies that often best way to "measure" a system's state is to simply observe its behavior over long periods of time and retroactively determine the state. Other methods of system identification employ this an approach. For instance, Takens' [15] method of embedding a single dimensional measurement into a higher dimensional space can provide a judgment of the underlying system's dimensionality. Crutchfield and Young extended this method to the computational understanding of systems by reconstructing the computational dynamics of the system from the sequence of boolean measurements.

The Counterexample

Two methods of assessing the computational ability of a system were described above. One relies on the identification of an isomorphism between the information processing (IP) states of a computational model and the observables of a system. The other method induces a computational model from a sequence of behaviors. The following example demonstrates how clustering of observable states can lead to the extraction of nonexistent deterministic states from recurrent networks. Recall the conditions under which two deterministic IP states are the same. Two states are the same if and only if they generate the same output responses for all possible future inputs. Clustering methods, as we shall see, cannot establish the existence of IP states when dealing with systems whose dynamics are sensitive to initial conditions. Consider a recurrent network with three state units in vector $S(t)$ and its dynamics specified by Equation 1 (with $a = 2$)[1]. The output of this network is a zero-thresholding of the first state unit. That is, when the current state is to the right of the boundary then the output is positive. The output is negative when it's to the left.

$$S(t+1) = \tanh\left(\begin{bmatrix} 0 & 0 & a & 0.5a \\ 0 & 0 & a & -0.5a \\ a & -a & 0 & -a \end{bmatrix} \cdot \begin{bmatrix} S(t) \\ 1 \end{bmatrix}\right) \tag{1}$$

Figure 1 illustrates what happens when the network iterates from a large set of initial conditions selected from within a small ball of radius 0.01. In one iteration these points migrate down to the lower corner of the state space, elongating along one dimension. After ten iterations the original ball shape is no longer visible. After twenty, the points are spread along a two dimensional sheet within state space. By fifty iterations we see the set of network states reaching their extent in state space. This behavior is ubiquitous in systems sensitive to initial conditions [16]. The assumption that the initial neighborhood defined an information processing state of a deterministic system is clearly invalidated by the vast set of unique output sequences that follow from this "state" without any assistance from inputs.

Iterated Function Systems

This section provides a brief overview of a new branch of mathematics known as iterated function theory [4]. While the limit behavior of single affine transformations (like $Ax + b$) can be determined analytically by examining eigenvalues of the transformations, iterated function theory can account for the limit behavior of systems of affine transformations.

---

1. $A \cdot \begin{bmatrix} B \\ 1 \end{bmatrix}$ indicates the inner product between A and the vector B with a 1 appended to that vector.

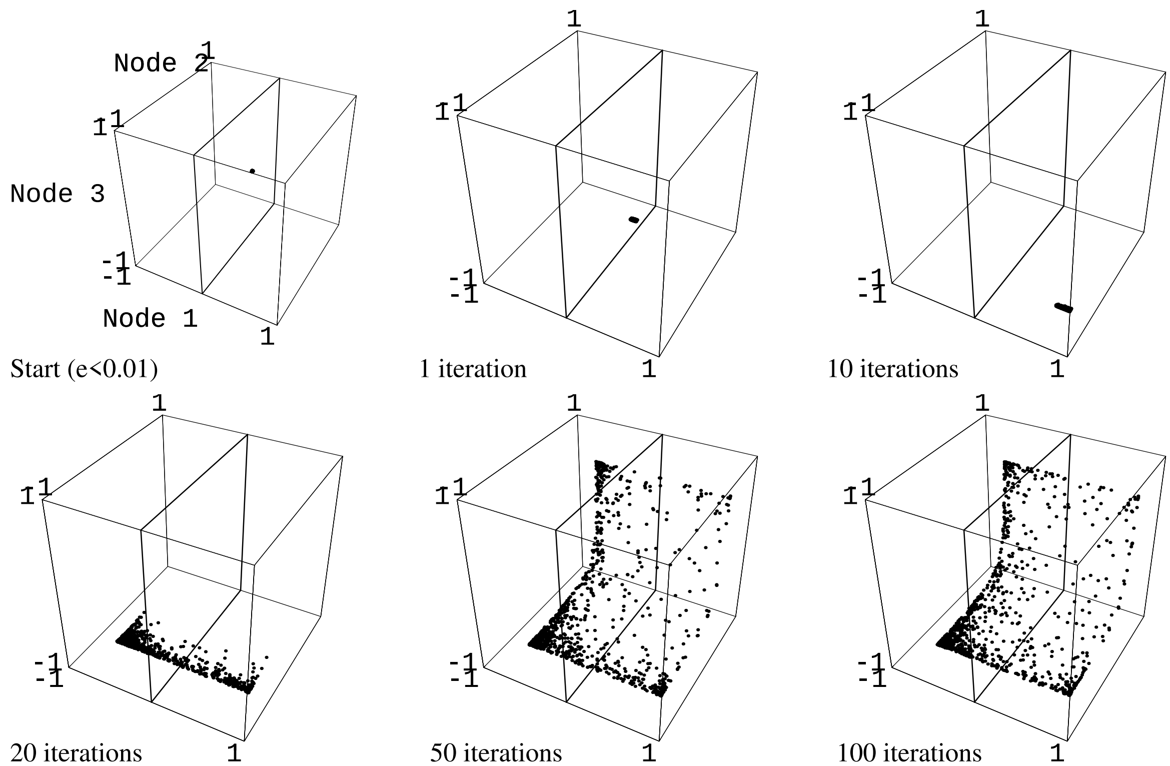

Figure 1: The state space of a recurrent network whose next state transitions are sensitive to initial conditions.

Basic Iterated Function Systems Theory

Formally, an iterated function system (IFS) is a metric space (X, d) and finite collection of affine transformations, $\{\omega_i | i \leq n, \omega_i: X \rightarrow X\}$. The attractor of an IFS is the set of points $A \subset X$ such that $\bigcup_{i=1}^{n} \omega_i(A) = A$. What makes IFSs so fascinating is that the limit behavior of a single transformation is just a point, yet the limit set over the union of the transformations can be extremely complex with recursive structure. IFSs are responsible for the structure seen in the Sierpinski triangle (Figure 2) and other fractals.

A metric space (X, d) is the combination of a set X and distance metric $d: X \times X \rightarrow \Re$. Consider the unit square and Euclidean distance function as our metric space. The three functions in Equation 2 map the unit square into three of the four quadrants of the unit square.

$$\omega_1((x, y)) = (0.5x, 0.5y + 0.5) \qquad \omega_2((x, y)) = (0.5x, 0.5y) \qquad \omega_3((x, y)) = (0.5x + 0.5, 0.5y) \qquad (2)$$

If we focus on a single transform, we find that there exists a limit of iterating this function starting with any point in the unit square. For instance, the fixed point of transform ω_1 of (0,1). This is an attracting fixed point since iteration draws other points closer to it over time.

A system of three transformation does not collapse into a single point. Rather, it has an attracting set with infinite recursive structure. A single transformation shrinks the entire limit set into a one-fourth sized copy of the original, while the union operation pastes the different copies together to form the original image again. This process is illustrated in Figure 3.

The attractor structure depends upon the transients of the transformation. In other words, the limit behavior of the individual transformations contributes little to the emergent shape of the attracting set. A simple example will demonstrate this fact. Each of the three transformations comprising the Sierpinski triangle IFS has an attracting limit point. These points, (0,1), (0,0), and (1,0), are also the limits of the transformations in Equation 3.The attractor of this set of transformations is the finite set {(0,1),(0,0),(1,0)}, which looks nothing like the Sierpinski triangle.

$$\omega_1((x, y)) = (0, 1) \qquad \omega_2((x, y)) = (0, 0) \qquad \omega_3((x, y)) = (1, 0) \qquad (3)$$

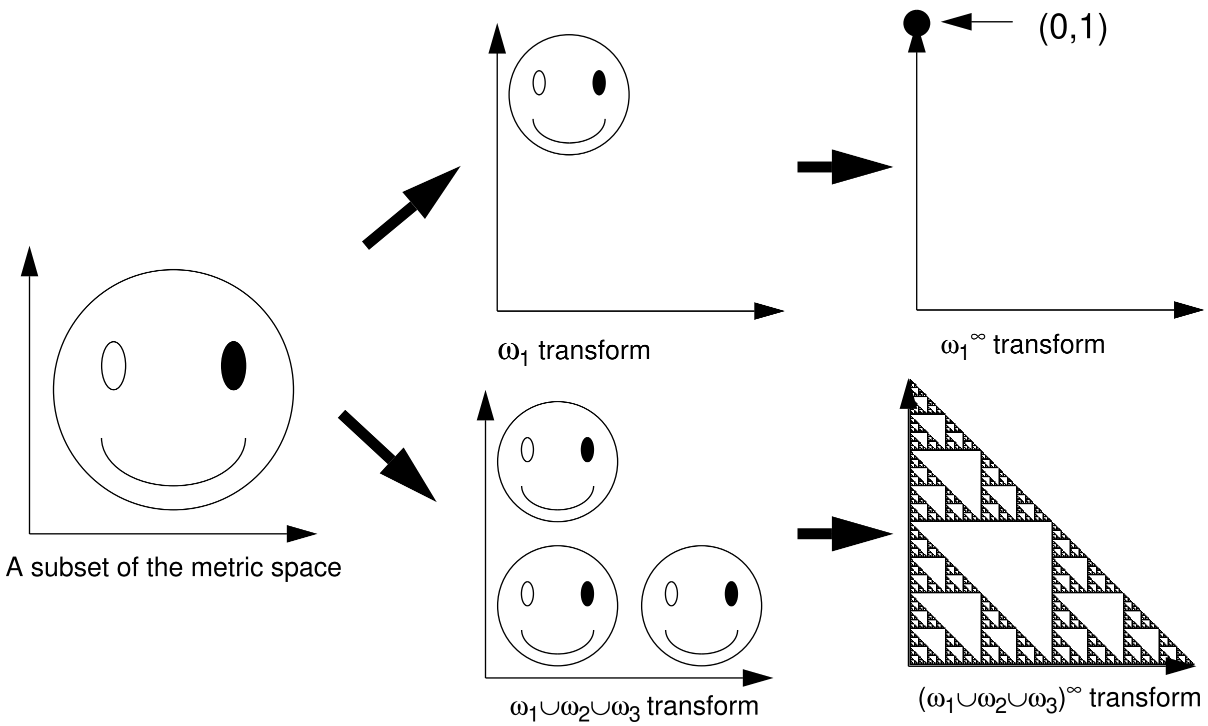

Figure 2: The difference between the limit of a single transformation (a point) and the limit of a collection of transformations (a Sierpinski triangle). The actual affine transformations are defined in Equation 2. The infinity superscript denotes the limit of an infinite sequence of function compositions.

An important concept for the understanding of iterated function systems is the Hausdorf space. The Hausdorf space of a complete metric space (X, d) is the set of all compact subsets of X, excluding the empty set. A compact set is a set that is both closed and totally bounded. The distance between two points in Hausdorf space is given by Equation 4. This metric measures the distance between two sets as the maximal distance between a point in A and the nearest point in B. Hausdorf space allows us to consider subsets of (X, d) as single points. It also lets us view the set of iterated transformations over the metric space as a single transformation in Hausdorf space. The theory laid out by Barnsley shows that IFSs composed of transformations which go to fixed points also produce fixed-point dynamics in Hausdorf space. For instance, it can be shown that the Sierpinski triangle is a point in $H(X)$ and that the Sierpinski IFS reduces the distance between any starting subset of X and the Sierpinski triangle.

$$h(d)(A, B) = \sup \{\inf \{d(x, y) | x \in A\} | y \in B\} \tag{4}$$

Random Iteration

An interesting side effect of the unique attractor for Barnsley's IFSs is that approximations to the attractor are easy to construct. Random iteration, also known as the chaos game, produces attractor approximations very rapidly. Take an initial point in the metric space and create a sequence of points by randomly selecting *one* IFS transformation at each iteration. The initial portion of the sequence is ignored as transients and the resulting union of the sequence remainder is a finite approximation to the attractor. It is easy to show that after each iteration the distance between the current point and the attractor geometrically decreases and approaches zero. The random transformation enforces coverage of the attractor, i.e. an infinite sequence of points generated this way will be dense in the attractor. The random iteration algorithm is sensitive to the method of selecting the transformations for each iteration. First, the probability of selecting a particular transformation can lead to an abnormal distribution of points approximating the attractor. Second, the selection generator must be ergodic, i. e. the selection must be independent of previous selections.

Language theory provides an alternative way of describing the random iteration algorithm. The sequence of transformations produced during random iteration can be compactly described with a stochastic finite state generator. The generator has probabilities associated with its transitions which produce variations in the sequence necessary for the production of the approximation. Without the probabilities, the generators produce a uniform distribution of sequences

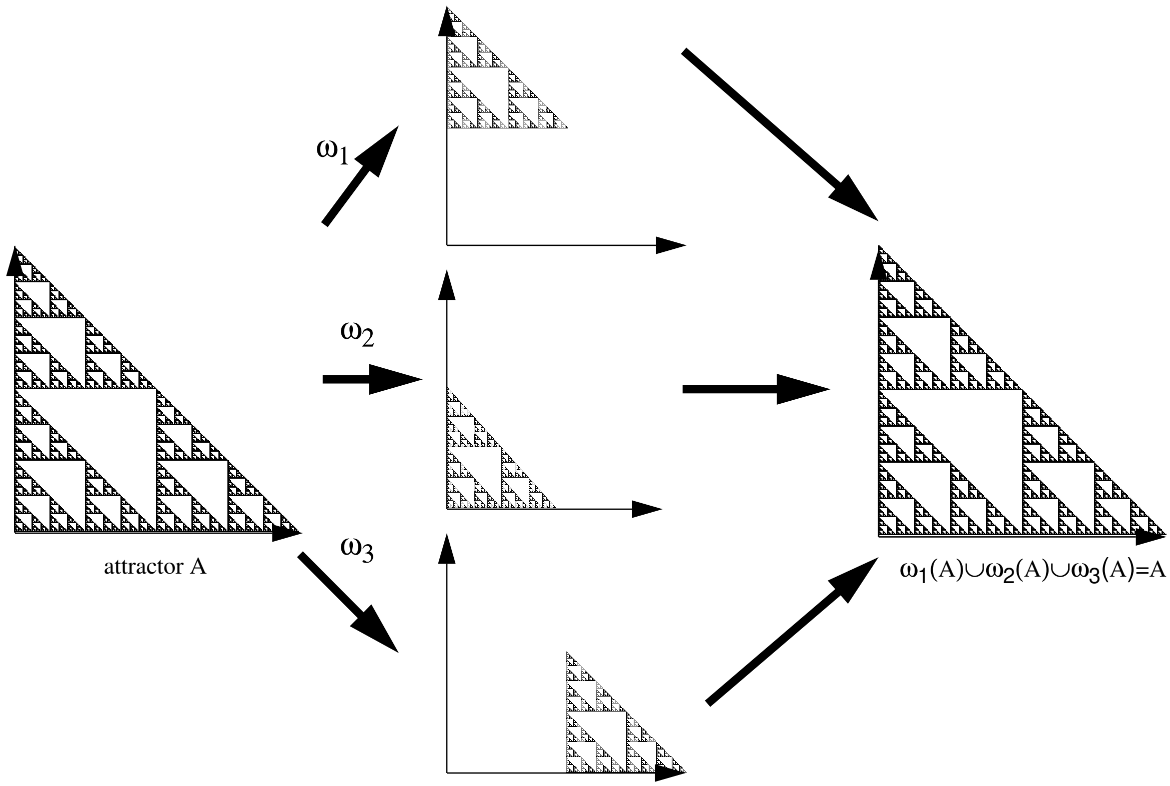

Figure 3: The individual transformations make three reduced copies of the attractor. Taking the union of the individual copies pastes together a the original attractor.

in Σ^{∞}, the set of all possible infinite sequences. Other "attractors" are possible by defining generators that produce subsets of Σ^{∞}.

Recurrent Networks as Iterated Function Systems

The behavior of recurrent networks in symbolic processing applications can be explained in terms of iterated function systems. This section provides a demonstration of the emergence of a complex recurrent network state space from the interaction of two simple state spaces. The recurrent network selected for this demonstration is a second order recurrent network know as the sequential cascaded network (SCN)[17]. An SCN can be described by two dynamic equations show in Equation 5.

$$O(t) = g\left(\left(W \cdot \begin{bmatrix} S(t) \\ 1 \end{bmatrix}\right) \cdot \begin{bmatrix} I(t) \\ 1 \end{bmatrix}\right) \qquad S(t+1) = g\left(\left(V \cdot \begin{bmatrix} S(t) \\ 1 \end{bmatrix}\right) \cdot \begin{bmatrix} I(t) \\ 1 \end{bmatrix}\right) \tag{5}$$

Where $O(t)$ is the output vector at time t, $S(t)$ is the state vector, $I(t)$ is the input vector, and W and V are three dimensional weight matrices describing state-to-state and state-to-output transformations performed by the SCN. The function g is the sigmoid function. Learning takes place in the SCN by adjusting the weight matrices W and V according to a modified form of back propagation. Training continues when the network successfully generates the desired outputs since overtraining forces the network through a qualitative change from a fixed point attractor to a limit cycle.

Pollack initially applied SCNs to the task of formal language recognition where the networks apparently induced regular languages. The internal states of these dynamical recognizers, however, were *not* finite. Dimensional analysis revealed a fractal dimension of 1.4 for the set of all possible internal states indicating an infinitely structured set [18].

In symbolic applications the set of predefined input vectors, Φ, is finite and we may write the dynamic in the

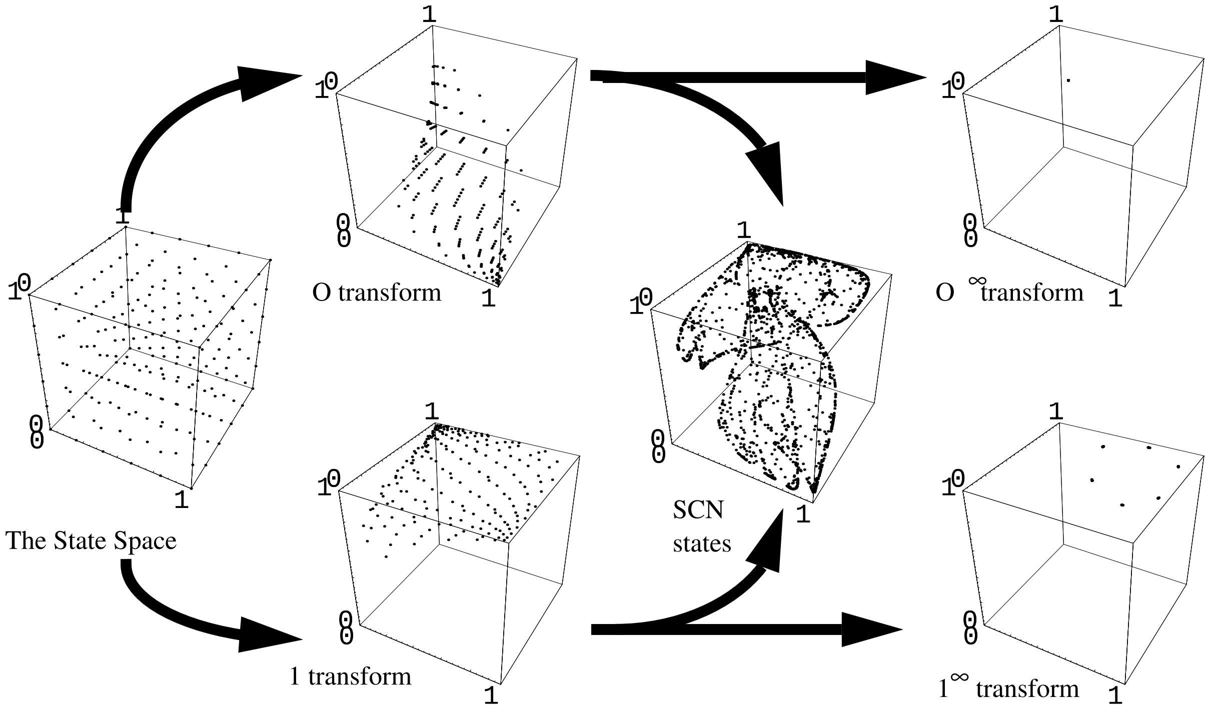

Figure 4: Dissection of the sequential cascaded network trained to accept 0*1*0*1*.

form shown in Equation 6.

$$O(t) = g(W_I \cdot \begin{bmatrix} S(t) \\ 1 \end{bmatrix}) \qquad S(t+1) = g(V_I \cdot \begin{bmatrix} S(t) \\ 1 \end{bmatrix}) \tag{6}$$

The parameters W_I and V_I are two dimensional matrices selected by input pattern $I \in \Phi$. If the transfer function g is linear and the set of inputs is finite, the SCN is mathematically equivalent to the set of affine transformations of an IFS. The SCN adds a sigmoid to “squash” the internal state into a small subregion, preventing activations from growing toward infinity and allowing non-fixed point attractors, such as limit cycles and chaos, in the individual transformations. To display an attractor formed by the selection of transformations, one can play the “chaos game” [4] by plotting a sequence of state points transformed by all sequences of transformations up to length ten. Pollack trained an SCN to recognize regular languages from a set of positive and negative examples. Figure 4 shows a state space dissection of the SCN trained on examples from the language 0*1*0*1*. While the network never learned the target language, it did exhibit the state space marked as “SCN states” in Figure 4. By breaking the SCN into two transformations, one for each input symbol, we can examine the individual transformations that produced the “magic mushroom.” The first iteration of the 0 transformation maps the state space onto a thin sheet along the diagonal while the first iteration of the 1 transformation pushes the state space to the ceiling with a one-sixth twist. In the limit, the 0 transformation produces a fixed point, while the 1 transformation gets closer and closer to this period 6 attractor.

The mushroom example demonstrates the existence of both fixed point and limit cycle attractors for individual inputs. Chaotic, or aperiodic, attractors are also possible in SCN state space. Figure 5 illustrates the effects of increasing parameter a on the asymptotic trajectories of the function f_a as shown in Equation 7.

$$f_a(x) = \tanh(a \tanh(ax + 0.5a) - a \tanh(ax - 0.5a) - a) \tag{7}$$

The resulting plot, known as a bifurcation diagram, captures the underlying structure of the period doubling phenomena. At the limits of period doubling (such as $a \approx 1.7$), the systems attractor is chaotic, thus implying infinite state behavior. The function f_a can be expressed as a SCN-like input selected transformation by spreading the iterated computation of over two time steps, as in Equation 1 where the third element of $S(t+1)$ is $f_a(x(t))$ and the first and second elements of $S(t)$ are $\tanh(ax + 0.5a)$ and $\tanh(ax - 0.5a)$.

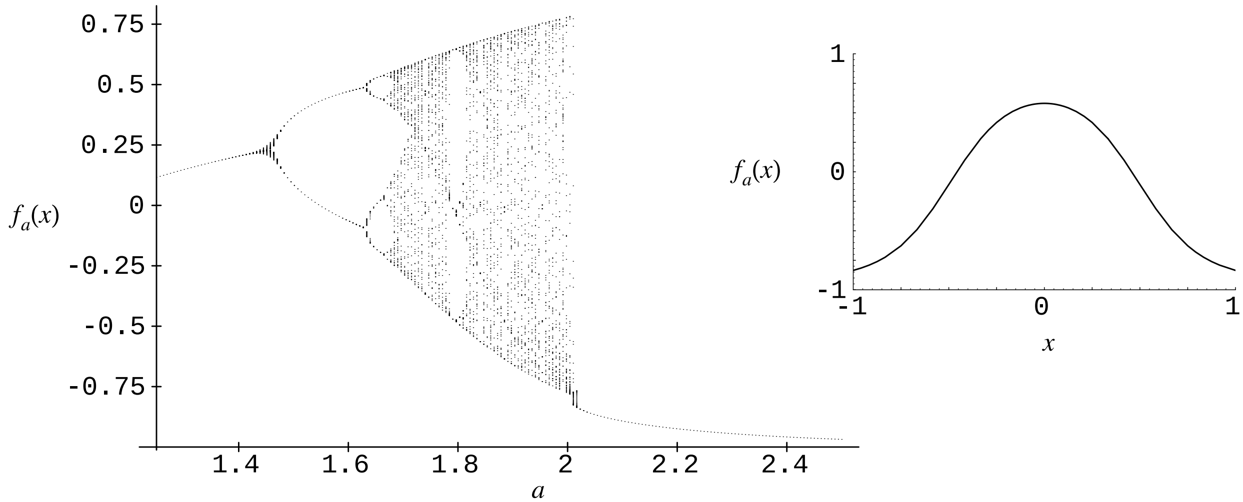

Figure 5: Sample mapping function and the bifurcation diagram that results from changes in control parameter a.

Discussion

Learning systems relying on recurrent networks carry a difficult burden; it is still unclear what these networks are processing, let alone what they learning. Many have assumed that the networks are learning to simulate finite state machines in their state dynamics and have begun to extract finite state machines from the networks' state transition dynamics [1, 2, 13]. The extraction methods employ various clustering techniques to partition the internal state space of the recurrent network into a finite number of regions corresponding to the states of a finite state automaton. As demonstrated in the first part of this paper, clustering of hidden unit activations, or recurrent network state space, provides incomplete information regarding the IP state of the network. IP states determine future behavior as well as encapsulate input history. The network's state transformations can exhibit sensitivity to initial conditions and generate disparate futures for state clusters of all sizes.

The second part of the paper presents IFS theory and shows how it can explain recurrent network state dynamics. By linking IFSs and recurrent networks, existing constraints on network dynamics independent of network models are now evident. By assuming a finite set of inputs, which is often the case in symbolic domains, one can describe recurrent network models as a finite collection of nonlinear state transformations.The interaction of several transforms produces complex state spaces with recursive structure. The limit behavior of the collection of transformations, and recurrent networks in symbolic applications, is more complex than the union of the individual transformations. An input driven recurrent network behaves like the random iteration algorithm. Infinite input sequence generates sequences of points dense in the state space attractor when the transformations are contractive. While the demonstration in this paper used the SCN, other models produce similar IFS-like behaviors as long as the network's input selects transformations [19].

The IFS approach also explains the phenomena of state clustering in recurrent networks. In [20], Serven-Schreiber *et al* report significant clustering in simple recurrent networks [21] both before and after training from the Reber grammar prediction task. A set of random transformations will normally reduce the volume of the recurrent networks state space, and place an upper bound on the distance between two transformed points. The upper bound has a significant effect on the clustering, especially when the transformations map to very small regions of state space. The task required that the clusters arranged themselves in a particular structure constrained by the single layer network generating predictions.

The utility of network states, however, is directly related to the complexity of the mechanism measuring the state. A computational system has three requirements. First, it should be input driven. Second, a computational system needs to dynamically change state. Third, it must generate output. While much has been written about the importance of state dynamics, the other two requirements are capable of elevating the behavioral complexity of the other two. Some systems produce perform transductions of periodic oscillations into complex signals. Different measurement devices can produce increases in both the number of apparent IP states and the observed complexity class of the system [22]. In other words, the recurrent network states are not IP states in of themselves; they require an appropriate context which can elevate them to IP-hood. This context consists of a set of input sequences and an observation method for generating outputs. While the recurrent network's state dynamics may be described as an IFS, any IP interpretation will involve a holistic combination of the set of possible inputs, the state dynamics, and the output generation mechanism of a network.

Acknowledgments

This work was supported by Office of Naval Research grant N00014-92-J-1195. The inspiration for this paper came from endless discussions with Jordan Pollack.

References

[1] C. L. Giles, C. B. Miller, D. Chen, G. Z. Sun, H. H. Chen and Y. C.Lee, "Extracting and Learning an Unknown Grammar with Recurrent Neural Networks," in *Advances in Neural Information Processing Systems 4*, 1992, pp. 317-324.

[2] R. L. Watrous and G. M. Kuhn, Induction of Finite-State Automata Using Second-Order Recurrent Networks, in *Advances in Neural Information Processing Systems 4*, 1992, pp. 309-316.

[3] C. L. Giles, S. Das and G. Sun, Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory, in *The Proceedings of the Fourteenth Annual Conference of the Cognitive Science Conference*, 1992, pp. 317-324.

[4] M. Barnsley, *Fractals Everywhere*, Academic Press, San Diego, CA, 1988.

[5] N. Chomsky, *Syntactic Structures*, Mounton & Co., The Hague, 1957.

[6] A. Newell and H. A. Simon, "Computer science as empirical inquiry: symbols and search," *Communications of the Association for Computing Machinery*, vol. 19, pp. 113-126, 1976.

[7] W. P. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[8] S. Franklin and M. Garzon, "Neural network implementation of Turing machines," in *The Proceedings of the Second Institute of Electrical and Electronics Engineers International Conference on Neural Networks*, 1988.

[9] J. B. Pollack, *On Connectionist Models of Natural Language Processing*, University of Illinois at Urbana-Champaign, Ph.D. thesis, 1987.

[10] H. T. Siegelmann and E. D. Sontag, "On the Computational Power of Neural Networks," Report SYCON-91-11, Rutgers Center for Systems and Control, 1991.

[11] B. MacLennan, "Field Computation in the Brain," Technical Report CS-92-174, University of Tennessee, 1992.

[12] W. R. Ashby, *An Introduction to Cybernetics*, Chapman and Hall, London, 1956.

[13] A. Cleeremans, D. Servan-Schreiber and J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural Computation*, vol. 1, pp. 372-381, 1989.

[14] J. Crutchfield and K. Young, "Computation at the Onset of Chaos," in *Entropy, Complexity, and the Physics of Information*, W. Zurek (ed.), Addison-Wesely, Reading, 1989.

[15] F. Takens, "Detecting strange attractors in turbulence", in *Lecture Notes in Mathematics 898*, Springer-Verlag, Berlin, 1981.

[16] J. Crutchfield, J. D. Farmer and N. Packard, "Chaos," *Scientific American*, vol. 255, pp. 46-57, 1987.

[17] J. B. Pollack, "The Induction of Dynamical Recognizers," *Machine Learning,* vol. 7,pp. 227-252, 1991.

[18] B. Mandelbrot, *The Fractal Geometry of Nature*, Freeman, San Francisco, CA, 1983.

[19] J. F. Kolen, Ph.D. thesis in progress, The Ohio State University.

[20] D. Servan-Schreiber, A. Cleeremans and J. L. McClelland, "Encoding Sequential Structure in Simple Recurrent Networks," Technical Report CMU-CS-183, 1988.

[21] J. Elman, "Finding structure in time," *Cognitive Science,* vol. 14, pp. 179-211, 1992.

[22] J. F. Kolen and J. B. Pollack, "The apparent computational complexity of physical systems," in *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, Laurence Erlbaum, 1993.