The Origin of Clusters in Recurrent Neural Network State Space

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

Abstract

Cluster analysis has been successfully applied to the problem understanding hidden unit representations in both feed-forward and recurrent neural networks. While the topological properties of feed-forward networks may support the use of cluster analysis, the results described within this paper suggest that applications to recurrent networks are not justified. This paper illustrates how clustering fails to provide useful insights into the underlying *task-dependent* information processing mechanism of recurrent networks. In this paper, I first demonstrate that randomly generated networks display a surprising amount of clustering *before* training. Then I explain that the clustering structure emerges, not in response to the task training, but because of the volume-reducing iterated mappings that comprise the commonly used recurrent neural networks models.

Introduction

The popularity of clustering analysis of neural network internal representations suggests that there is some merit to this approach. This technique has been successfully applied to the problem of understanding hidden unit representations in both feed-forward and recurrent networks. The application of hierarchical clustering to this particular problem was originally suggested to Sejnowski and Rosenberg by Stephan Hanson (as reported in (Hanson & Burr, 1989)). Clustering of internal representations of the NETtalk system provided evidence of vowel/consonant discrimination in the hidden units (Sejnowski & Rosenberg, 1987). Elman also used this tool to illustrate the internal state clustering in his simple recurrent network (Elman, 1990). In this tradition, many studies have reported the emergence of various "concept" or "state" clusters during recognition and production tasks (e.g., (Servan-Schreiber et al., 1988; Cleeremans et al., 1989; Pollack, 1990; Elman, 1992; Meeden et al., 1993a; Cummins & Port, 1994; Crucianu, 1994)). State-vector clustering algorithms serve as the base mechanism for the extraction of finite state machine descriptions from recurrent networks (Giles et al., 1992; Watrous & Kuhn, 1992). While the topological properties of feed-forward networks may support the use of cluster analysis, the results described within this paper illustrate an application of clustering which fails to provide any useful insights into the underlying *task-dependent* information processing mechanism of recurrent networks.

The assumption confronted below is that the neighborhood relationships between internal state vectors of a recurrent network will help us understand the processing performed by the network at a cognitive level. In fact, we can explain the clustering phenomena without appealing to information processing: the key is the theory of iterated function systems (IFS's). IFS theory also helped explain why recurrent networks often produce infinite state spaces (Kolen, 1994c) and why finite state machine extraction techniques can produce finite state descriptions with high, but illusionary, complexity (Kolen, 1994b).

Iterated Function Systems

The foundational work was originally developed by Barnsley (1988) as a method of describing the limit behavior of systems of transformations. The limit behavior of a single linear or affine transformation can be determined by examining eigenvalues of the transformations. The trajectories can either be fixed points or limit cycles.[1] While the effects iterating of linear systems have been fully mapped out at this time, only recently did anyone consider the case of multiple affine transformations in parallel, i.e., $f(x) = \bigcup_{i=1}^{n} \omega_i(x)$. Such systems have now been shown to be a generalization of Cantor's "middle third" sets.What makes IFSs so fascinating is that the limit behavior of a single transformation is just a point, the limit set over the union of the transformations can be extremely complex with recursive structure. A review of IFS theory as it applies to recurrent networks appears in (Kolen, 1994a; Kolen, 1994c). Suffice it to say that a recurrent network in an environment consisting of a finite set of input vectors will behave as a nonlinear IFS.

The notion of IFS address is important for the current discussion. Every IFS attractor, the limit behavior of the composite mapping, has an addressing scheme defined by the set of transformations (Equation 1).

$$\begin{aligned} \omega_1((x, y)) &= (0.5x, 0.5y + 0.5) \\ \omega_2((x, y)) &= (0.5x, 0.5y) \\ \omega_3((x, y)) &= (0.5x + 0.5, 0.5y) \end{aligned} \qquad \text{(Eqn 1)}$$

An address of a point on an attractor is the infinite

1. Another regime exists, known as quasiperiodicity, and occurs in iterated maps with limit cycles whose angle of rotation is irrational.

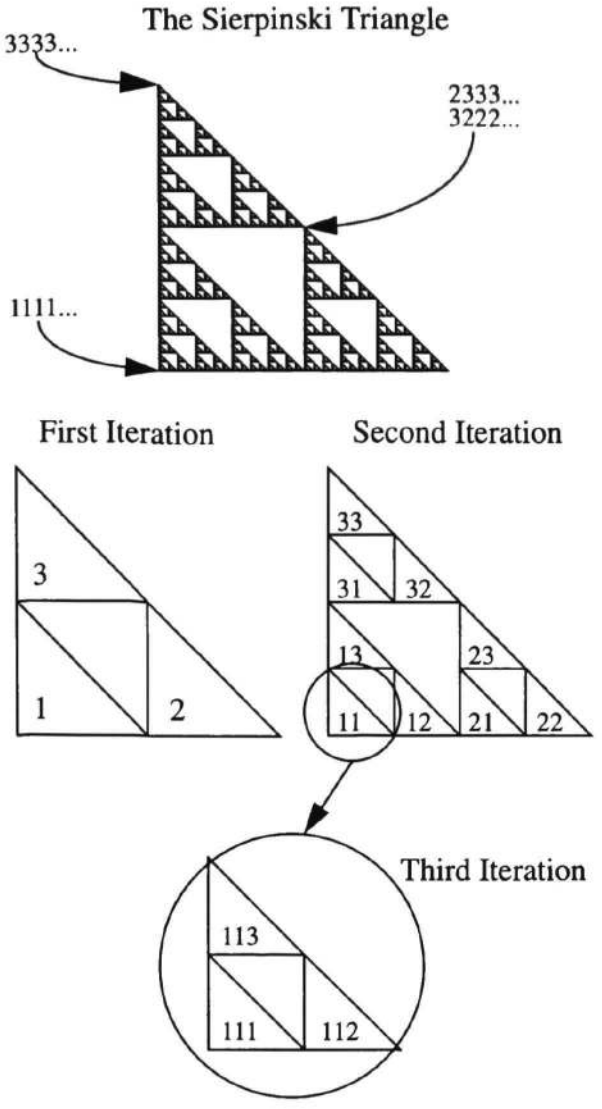

Figure 1: The Sierpinski Triange and an addressing scheme for points on the attractor. Notice that some points can have multiple addresses because the transformations touch or overlap.

sequence of transformations whose limit is that point when the starting point is the entire space. In the case of the Sierpinski triangle, the first entry of the address of all the points in the upper left corner are the same. This process continues recursively within each mapping of the state space. Because the IFS relies on contractive mappings, the regions shrink with each transformation application, the limit of which is our target point. Figure 1 illustrates the addressing scheme for the Sierpinski triangle. The numbers refer to the transformation number. Starting with the final attractor (top of Figure 1), each transformation copies the original image into three regions (see First Iteration). Each copy has been labeled the transform that placed it. This is the address. On the second iteration, the copying process continues. This time, however, the transform number is prepend to any existing label. This labeling process will continue indefinitely as

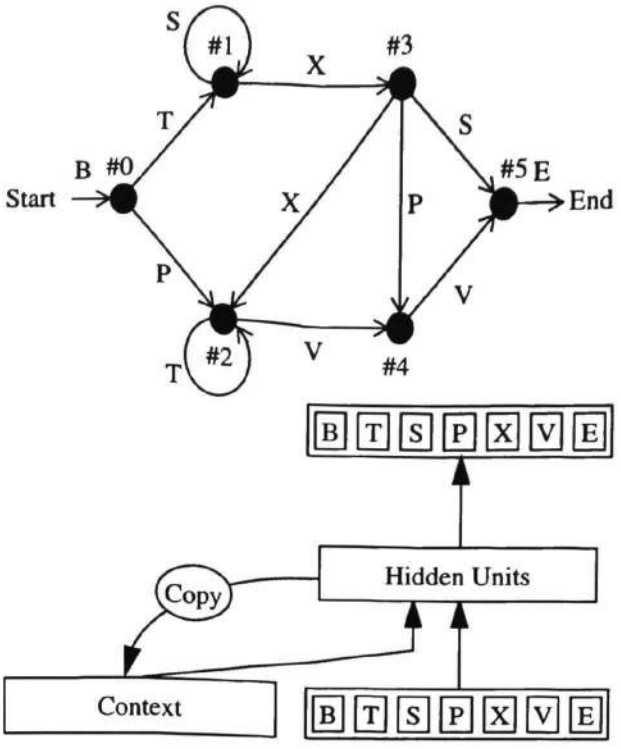

Figure 2: The Reber grammar and the simple recurrent network described in (Servan-Schreiber et al., 1988).

each region becomes smaller and smaller. In the limit, the regions will be points. At this time, the address will be an infinite sequence of transformation labels.

Clustering the Reber Grammar in Arbitrary Networks

The IFS address is the reverse of the input alphabet to a recurrent network. The first element of the address was the most recent input symbol. The second element of the address was next recent input. If the transformations in the recurrent network are contractive, one could predict clustering on recency solely from the mathematics of the state transformations and independent from the learning task.

To demonstrate the validity of this claim, I will refer to experimental findings encountered during the grammar learning (Servan-Schreiber et al., 1988; Cleeremans et al., 1989). In their paper, they took a simple recurrent network (SRN) and trained it in the prediction task for the Reber grammar. The finite state machine underlying the Reber grammar and the network used to predict it is displayed in Figure 2. The rationale behind selecting this grammar over others is its historical position of being used in several psychological experiments on implicit learning (Reber, 1967). The goal of the network's task is to predict the next symbol, or symbols, given the current context. The network received input in the form of a one-in-seven encoding of the seven characters of the Reber alphabet. The output consisted of seven units in a similar encoding scheme. Recall that the dynamics of the SRN look like

Equation 2, where g is the sigmoid function and W is the weight matrix.

$$S^{(t+1)} = g(W \cdot \begin{bmatrix} S^{(t)} \\ I^{(t)} \\ 1 \end{bmatrix}) \qquad \text{(Eqn 2)}$$

In (Kolen, 1994c) I demonstrate that this dynamic can be rewritten given that the set of input vectors is finite. For each input vector, I, there exists a weight matrix W_I that the dynamics of Equation 2 will be reduced to Equation 3.

$$S^{(t+1)} = g(W_I \cdot \begin{bmatrix} S^{(t)} \\ 1 \end{bmatrix}) \qquad \text{(Eqn 3)}$$

When an input appears, the correct weight matrix is selected and applied to the current state. This particular implementation is indistinguishable from iteration of Equation 2. In light of this, I will generate a set of indexed transformations and perform a cluster analysis of the states encountered during the processing of the Reber grammar. Table 1 lists the parameters of the five transformations for, one for each symbol in the grammar. The parameters of the transformation were selected from a normal distribution with a mean of zero and standard deviation of one.[2] Recall that different recurrent network architectures have different IFS interpretations. To implement the behavior of a SRN, the individual transformations must only differ in their the additive factors. In this demonstration, I have constructed SRN-like transformations with a representation dimension of two (Equation 4).

$$S^{(t+1)} = g(\begin{bmatrix} a & b \\ c & d \end{bmatrix} S^{(t)} + \begin{bmatrix} e \\ f \end{bmatrix}) \qquad \text{(Eqn 4)}$$

The motivation for the number of hidden was presentational: it's easier to view planar transformations. Similar arguments will hold for higher dimensional representation spaces. Table 1 lists the parameters of the five transformations, one for each symbol in the grammar (excluding **B** and **E**). Because the SRN architecture precludes input modification of the multiplicative parameters, the first four columns of Table 1 (corresponding to the parameters a, b, c, and d, of Equation 4) are equal across each transformation. The transformation parameters were selected from a normal distribution with a mean of zero and standard deviation of one.

The five plots labeled t, p, s, x, and v, in Figure 3 show how the individual transforms map a 15x15 grid of equally spaced points covering the unit plane back onto the unit plane. Also in that figure is a chaos game[3] exploration of the state space (Kolen, 1994c). Each transformation had 0.2 probability of selection. The first 100 points were ignored as transients, and the last 500 points were plotted. This plot provides an approximate picture of the state representations of Σ^*.

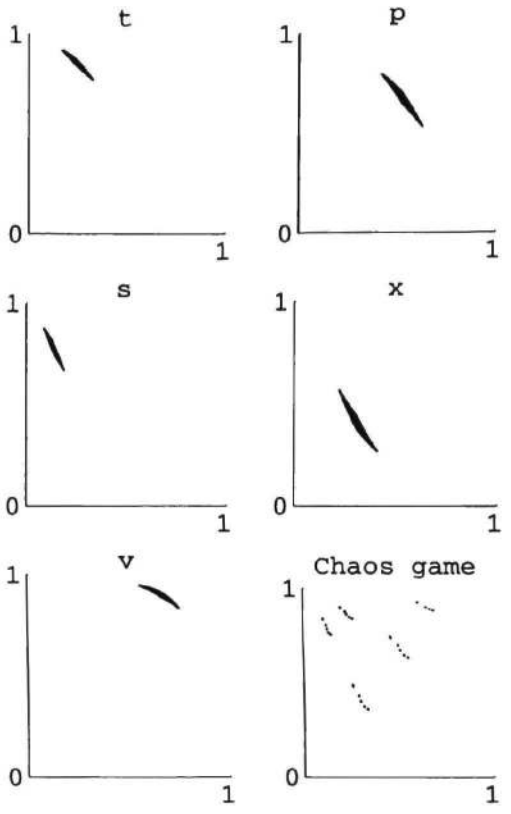

Figure 3: The SRN transformations and hidden unit representations. The axes measure the activations of state nodes one and two.

Figure 4 illustrates the SRN state representation of all Reber strings of length eight or less. The graphs labeled #1, #2, #3, #4, and #5 report the activation of the state vector when the generator was in the corresponding state. The final graph, labeled "All Reber States", is the union of the other five state graphs. Notice the similarity between the attractor approximation and the set of valid Reber states. Any decision mechanism using hyperplanes will have a difficult time differentiating between Reber and non-Reber strings solely on the basis of the state activation.

In an earlier report, Servan-Schreiber et al. (1988) produced a cluster diagram of hidden unit activations before any training has occurred. This diagram clearly shows that the internal representations of the network were clustering by most recent symbol. Cleeremans' et al went on to claim that the clustering of hidden unit representations was a product of training the network in the prediction task. These representations, in their eyes, captured regularities in the previous symbols. I disagree with this explanation. Since the Reber grammar states are uniquely determined by the last two symbols, the clustering comes for free once the transforms no longer overlap. Figure 4 shows the state vectors for all strings up to length 8 which still lie on the

2. No learning is taking place, any emerging structure in the state space will be independent of any task.
3. The chaos game creates a sequence of points from a seed point by selecting a random sequence of IFS transformations, applying it to the current point (Barnsley, 1988). The sequence of current points is a useful approximation to the attractor by ignoring the transient points at the beginning of the sequence.

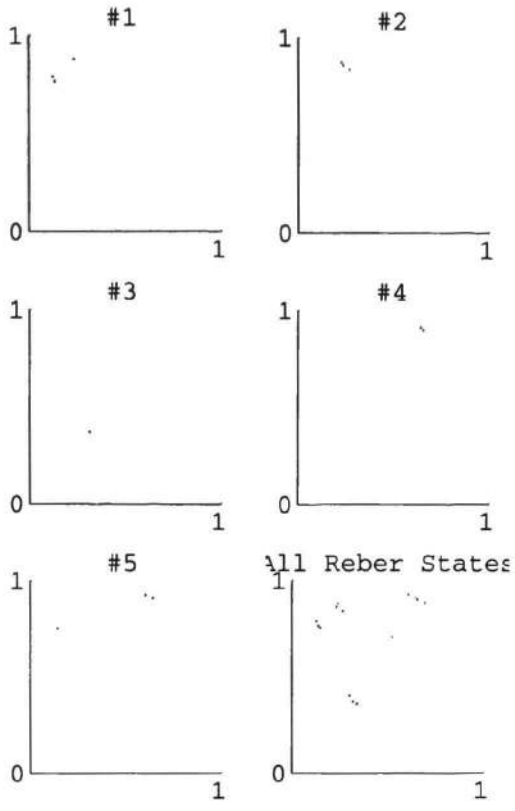

Figure 4: The hidden unit representations of all Reber strings of length eight or less. The axes measure the activations of state nodes one and two

Reber state graph. As you can see, the states clump together well. A cluster diagram of these vectors, in Figure 5, emphasizes this observation. Since most learning systems start with small weight assignments, one can conclude that much of the learning time is spent waiting for the recurrent state transformation to separate and be able to encode input symbol histories

The mechanism producing state space clusters in SRNs is not unique to this particular recurrent network formalism. Other recurrent networks display similar volume-reducing transformations. For instance, to implement the behavior of a SRN, the individual transformations must only differ in their the additive factors. Sequential Cascaded Networks (Pollack, 1991), on the other hand, can have independent transformations. An SCN is a second-order version of the SRN. A sample set of transformations appears in Table 2. An analysis, like the one described above, produced the state transformations appear in Figure 6 and the Reber states plotted in Figure 7. (A complete analysis is detailed in (Kolen, 1994a).) Even though the multiplicative parameters differ for each transformation, creating more variability in the shapes and orientations of the transformations, the states always display a hierarchical organization paralleling the IFS addressing scheme. These arguments easily extend to Jordan networks and any other discrete time recurrent neural networks with a finite set of input vectors.

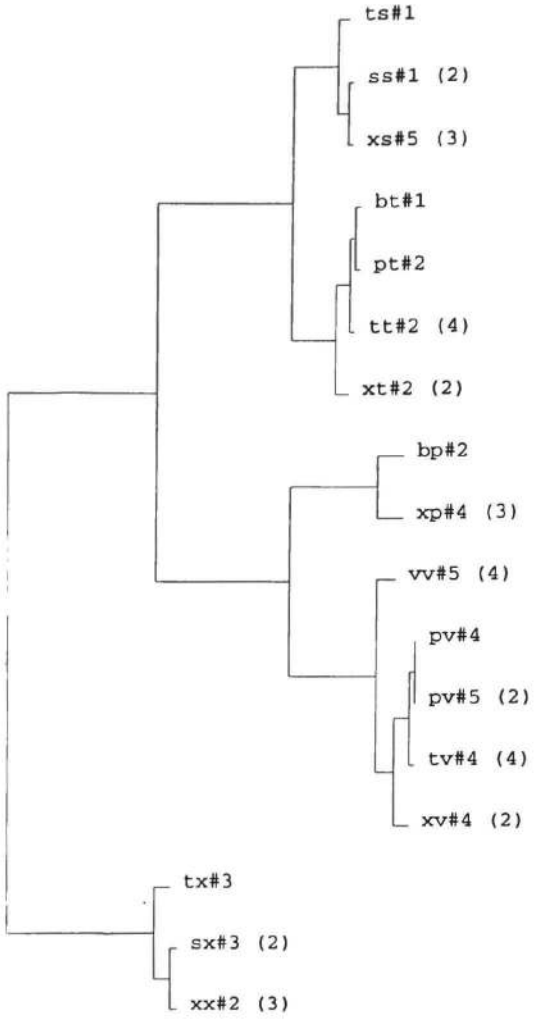

Figure 5: The resulting cluster diagram of all Reber strings of length eight or less on a random set of SRN transformations. The tree has been simplified, the numbers in parentheses is the total number of strings at that node with the same last two symbols.

Conclusion

Neural information processing approach to cognitive science and artificial intelligence problems, such as formal language learning (Gold, 1969), involves the use of recurrent networks that embody the internal state mechanisms underlying automata models (Pollack, 1991; Elman, 1992; Giles et al., 1992; Servan-Schreiber et al., 1988; Watrous & Kuhn, 1992). Unlike traditional automata- and grammar-based approaches, learning systems relying on recurrent networks carry a difficult burden: it is still unclear what these networks are processing, let alone what they are learning.

The IFS approach explains the phenomena of state clustering in recurrent networks. Servan-Schreiber *et al* reported significant clustering in simple recurrent networks both before and after training from the Reber gram-

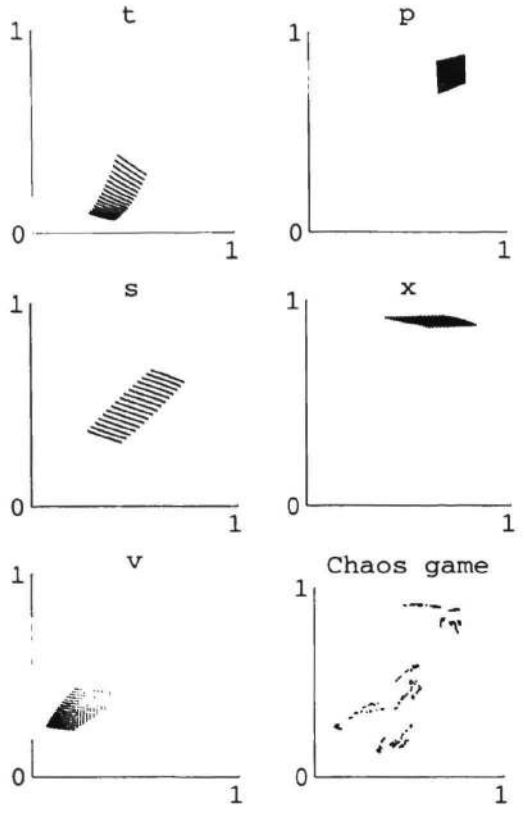

Figure 6: The Sequential Cascaded Network (SCN) transformations and hidden unit representations. The axes measure the activations of state nodes one and two.

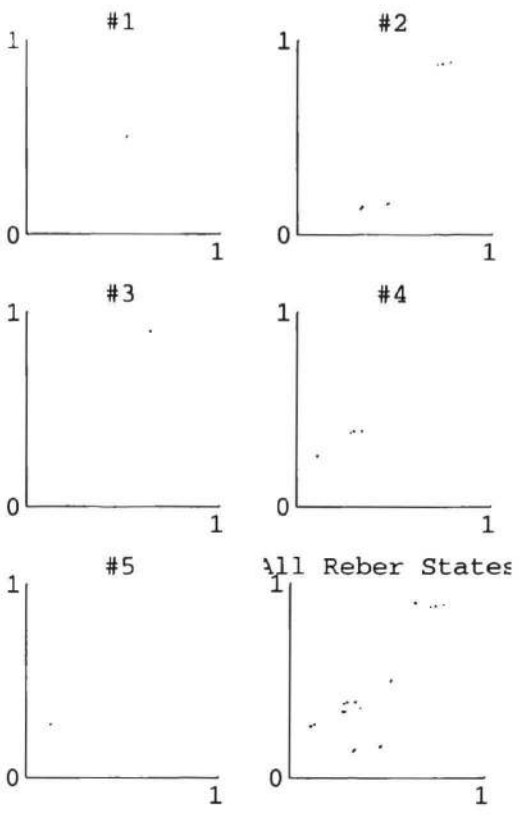

Figure 7: The hidden unit representations of all Reber strings of length eight or less in the SCN. The axes measure the activations of state nodes one and two

mar 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 prediction task requires that the network arrange it state clusters to satisfy constraints imposed by the single layer network generating predictions.

Thus, the single most cited "discovery" attributed to recurrent network training is, in fact, a property independent of its training. It appears that the network is adjusting its observation of the internal state, the state-to-output mapping, to accommodate the task. Such observation shifts are capable of inducing a wide variety of behavioral complexities (Kolen, 1993). Given that the recurrent network states cluster themselves, it is always possible to construct a disjunctive output function for a randomly generated network that arbitrarily labels these regions with the correct state labeling.

In order to understand the behavior of recurrent networks, these devices should be regarded as dynamical systems (Kolen, 1994a). In particular, most common recurrent networks are actually iterated mappings, nonlinear versions of Barnsley's iterated function systems (Barnsley, 1988). While automata also fall into this class, they are a specialization of dynamical systems, namely discrete time and state systems. Unfortunately, information processing abstractions are only applicable within this domain and do not make any sense in the broader domains of continuous time or continuous space dynamical systems.

Acknowledgments

This work was supported by the Office of Naval Research through grant number N00014-92-J-1195.

References

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

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

Crucianu, M. (1994). Looking for structured representations in recurrent networks. In *Proceedings of the 1993 Connectionist Models Summer School.* American Elsevier. 170-177.

Cummins, F. & Port, R. F. (1994). On the treatment of time in recurrent neural networks. In *Proceedings of the 1993 Connectionist Models Summer School.* American Elsevier. 211-218.

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

Elman, J. L. (1992). Distributed Representations, Simple Recurrent Networks, and Grammatical Structure. *Machine Learning, 7.*

Giles, C. L., Miller, C. B., Chen, D., Sun, G. Z., Chen, H. H. & Lee, Y. C. (1992). Extracting and Learning an Unknown Grammar with Recurrent Neural Networks. In J. E. Moody, Steven J. Hanson & Richard P. Lippman, (Eds.), *Advances in Neural Information Processing Systems 4.* Morgan Kaufman.

Gold, E. M. (1969). Language identification in the limit. *Information and Control, 10,* 372-381.

Hanson, S. J. & Burr, D. J. (1989). What connectionist models learn: learning and representation in connectionist networks. *Behavioral and Brain Sciences,13,*471-518.

Kolen, J. F. & Pollack, J. B. (1993). The apparent computational complexity of physical systems. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Earlbaum.

Kolen, J. F. (1994a). *Exploring the Computational Capabilities of Recurrent Neural Networks.* Ph.D. thesis. Department of Computer and Information Science. The Ohio State University.

Kolen, J. F. (1994b). Fool's Gold: Extracting Finite State Machines From Recurrent Networks Dynamics. In J. D. Cowan, G. Tesauro & J. Alspector, (Eds.), *Advances in Neural Information Processing Systems 6.* Morgan Kaufman.

Kolen, J. F. (1994c). Recurrent networks: State machines or iterated function systems? In *Proceedings of the 1993 Connectionist Models Summer School.* American Elsevier. 203-210.

Meeden, L., McGraw, G. & Blank, D. (1993). Emergent Control and Planning in an Autonomous Vehicle. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Earlbaum. 735-740.

Pollack, J. B. (1990). Recursive autoassociative memories. *Artificial Intelligence, 46,* 77-105.

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning, 7,* 227-252.

Reber, A. S. (1967). Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior, 5,* 855-863.

Sejnowski, T. J. & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce english text. *Complex Systems, 1,* 145-168.

Servan-Schreiber, D., Cleeremans, A. & McClelland, J. L. (1988). Encoding sequential structure in simple recurrent networks. UCSD Technical Report.

Watrous, R. L. & Kuhn, G. M. (1992). Induction of Finite-State Automata Using Second-Order Recurrent Networks. In J. E. Moody, S. J. Hanson & R. P. Lippman, (Eds.), *Advances in Neural Information Processing Systems 4.* Morgan Kaufman.

Table 1: The SRN Transformations

SYM	a	b	c	d	e	f
t	-0.444983	-0.433067	0.759095	0.492625	-0.719507	1.23284
p	-0.444983	-0.433067	0.759095	0.492625	0.528296	0.121272
s	-0.444983	-0.433067	0.759095	0.492625	-1.44764	0.700865
x	-0.444983	-0.433067	0.759095	0.492625	-0.369454	-1.00214
v	-0.444983	-0.433067	0.759095	0.492625	1.11458	1.59991

Table 2: The SCN Transformations

SYM	a	b	c	d	e	f
t	-0.633273	-0.558169	-1.73999	0.431231	0.259375	-0.89197
p	-0.025588	-0.677765	0.962798	-0.277900	1.317470	1.12233
s	1.300490	-0.697644	1.21811	0.243118	-0.262879	-0.76881
x	1.171490	0.866640	0.111505	-0.496449	-0.464076	2.37686
v	-1.220900	-1.254000	0.09564	-0.82854	-0.105562	-0.36281