%I didnt finish the addendum yet but %but added some stuff up front and %played with all table and figure legends %figures to follow... Jordan

delim $$ gsize 12

The Induction of Dynamical Recognizers

Jordan B. Pollack

Laboratory for AI Research &
Computer & Information Science Department
The Ohio State University
2036 Neil Avenue
Columbus, OH 43210
(614) 292-4890
pollack@cis.ohio-state.edu

ABSTRACT

A higher order recurrent neural network architecture learns to recognize and generate languages after being "trained" on categorized exemplars. Studying these networks from the perspective of dynamical systems yields two interesting discoveries: First, a longitudinal examination of the learning process illustrates a new form of mechanical inference: Induction by phase transition. A small weight adjustment causes a "bifurcation" in the limit behavior of the network. This phase transition corresponds to the onset of the network's capacity for generalizing to arbitrary-length strings. Second, a study of the automata resulting from the acquisition of previously published training sets indicates that while the architecture is *not* guaranteed to find a minimal finite automaton consistent with the given exemplars, which is an NP-Hard problem, the architecture does appear capable of generating non-regular languages by exploiting fractal and chaotic dynamics. I end the paper with a hypothesis relating linguistic generative capacity to the behavioral regimes of non-linear dynamical systems.

Table 1

center box; c|c l|l . ACCEPT REJECT _ 1 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 1 0 0 1 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 1

1. Introduction

Consider the two categories of binary strings in Table 1. After brief study, a human or machine learner might decide to characterize the "accept" strings as those containing an odd number of 1's and the "reject" strings as those containing an even number of 1's.

The language acquisition problem has been around for a long time. In its narrowest formulation, it is a version of the inductive inference or "theory from data" problem for syntax: Discover a compact mathematical description of string acceptability (which generalizes) from a finite presentation of examples. In its broadest formulation it involves accounting for the psychogical and linguistic facts of native language acquisition by human children, or even the acquisition of language itself by *Homo Sapiens* through natural selection (Lieberman, 1984; Pinker & Bloom, 1990)

The problem has become specialized across many scientific disciplines, and there is a voluminous literature. Mathematical and computational theorists are concerned with the basic questions and definitions of language learning (Gold, 1967), with understanding the complexity of the problem (Angluin, 1978; Gold, 1978), or with good algorithms (Berwick, 1985; Rivest & Schapire, 1987). An excellent survey of this approach to the problem has been written by (Angluin & Smith, 1983). Linguists are concerned with grammatical frameworks which can adequately explain the basic fact that children acquire their language (Chomsky, 1965; Wexler & Culicover, 1980), while psychologists and psycholinguists are concerned, in detail, with how an acquisition mechanism substantiates and predicts empirically testable phenomena of child language acquisition. (MacWhinney, 1987; Pinker, 1984).

My goals are much more limited than either the best algorithm or the most precise psychological model; in fact I scrupulously avoid any strong claims of algorithmic efficiency, or of neural or psychological plausibility for this initial work. I take as a central research question for connectionism:

> *How could a neural computational system, with its slowly-changing structure, numeric calculations, and iterative processes,* ever come to possess *linguistic generative capacity, which seems to require dynamic representations, symbolic computation, and recursive processes?*

Although a rigorous theory may take some time to develop, the work I report in this paper does address this question. I expose a recurrent higher order back-propagation network to both positive and negative examples of boolean strings, and find that although the network does *not* converge on the minimal-description finite state automaton for the data (which is NP-Hard), it does induction in a novel and interesting fashion, and searches through a hypothesis space which, theoretically, is not constrained to machines of finite state.

These results are of import to many related neural models currently under development, e.g. (Elman, 1990; Giles et al., 1990; Servan-Schreiber et al., 1989), and ultimately relates to the question of how linguistic generative capacity can arise in nature.

Generative capacity is a measure of a formal system's (and thus a natural system's) ability to generate, recognize, or represent languages in the limit, where a language here is defined as an infinite set of finite-length strings composed from a fixed vocabulary. The beautiful results derived from Chomsky's (1956) insights in the earliest years of modern computer science and linguistics research are that certain mechanisms, as well as certain rewrite systems (grammars), are tightly correlated with certain classes of languages. Although we now know of many other language classes, the 4 main language types - regular, context-free, context-sensitive, and recursive, form an inclusive hierarchy which is precisely matched to a set of computational models.

Furthermore, the existence of certain phenomena of syntactic structure in English, such as center embedding ("the rat the cat the dog chased bit died") or crossed serial dependencies (e.g. the "respectively" construction) which could not be generated by regular or context-free languages, respectively, indicated that the human mind was operating at a higher level of computational behavior.

Thus, the issue of generative capacity is a serious concern for Connectionism because since 1956 it has been firmly established that regular languages are inadequate to explain (or at least parsimoniously describe) the syntactic structures of natural languages. To the extent that iterated systems like neural networks, are equivalent in their generative capacity to Markov chains and finite-state machines, they should not even be under consideration as appropriate models for natural language.

Of necessity, I will make use of the terminology of non-linear dynamical systems for the remainder of this article. This terminology is not (yet) a common language to most computer and cognitive scientists and thus warrants an introduction. The view of neural networks as non-linear dynamical systems is commonly held by the physicists who have helped to define the modern field of neural networks (Hopfield, 1982; Smolensky, 1986), although complex dynamics have generally been suppressed in favor of more tractable convergence (limit point) dynamics. But chaotic behavior has shown up repeatedly in studies of neural networks (Derrida & Meir, 1988; Huberman & Hogg, 1987; Kolen & Pollack, 1990; Kurten, 1987; van der Maas et al., 1990), and a few scientists have begun to explore how this dynamical complexity could be exploited for useful purposes, e.g. (Hendin et al., 1991; Pollack, 1989; Skarda & Freeman, 1987).

In short, a discrete dynamical system is just an iterative computation. Starting in some "initial condition" or state, the next state is computed as a mathematical function of the current state, sometimes involving parameters and/or input or noise from an environment. Rather than studying the *function* of the computations, much of the work in this field has been concerned with explaining universal temporal *behaviors*. Indeed, iterative systems have some interesting properties: Their behavior in the limit reaches either a steady state (limit point), an oscillation (limit cycle), or an aperiodic instability (chaos). In terms of computer programs, these three "regimes" correspond, respectively, to those programs which halt, those which have simple repetitive loops, and those which have more "creative" infinite loops, such as broken self-modifying codes, an area of mechanical behavior which has not been extensively studied. When the state spaces of dynamical systems are plotted, these three regimes have characteristic figures called "attractors": Limit points show up as "point attractors", limit cycles as "periodic attractors", and chaos as "strange attractors", which usually have a "fractal" nature. Small changes in controlling parameters can lead through "phase transitions" to these qualitatively different behavioral regimes; a "bifurcation" is a change in the periodicity of the limit behavior of a system, and the route from steady-state to periodic to aperiodic behavior follows a universal pattern. Finally, one of the characteristics of chaotic systems is that they can be very sensitive to initial conditions, and a slight change in the initial condition can lead to radically different outcomes. Further details can be found in articles and books on the field, e.g. (Crutchfield et al., 1986; Devaney, 1987; Gleick, 1987; Grebogi et al., 1987)

2. Automata, Recurrent Networks, and Dynamical Recognizers

I should make it clear from the outset that the problem of inducing *some* recognizer for a finite set of examples is "easy", as there are an infinite number of regular languages which account for a finite sample, and an infinite number of automata for each language. The difficult problem has always been finding the "minimal description," and no solution is asymptotically much better than "learning by enumeration" -- brute-force searching of all automata in order of ascending complexity. Another difficult issue is the determination of grammatical class. Because a finite set of examples does not give any clue as to the complexity class of the source language, one apparently must find the most parsimonious regular grammar, context-free grammar, context-sensitive grammar, etc., and compare them. Quite a formidable challenge for a problem-solver!

Thus, almost all language acquisition work has been done with an inductive bias of presupposing some grammatical framework as the hypothesis space. Most have attacked the problem of inducing finite-state recognizers for regular languages, e.g., (Feldman, 1972; Tomita, 1982).

A Finite State Recognizer is a quadruple $\{ Q, \Sigma, \delta, F \}$, where Q is a set of states (q_0 denotes the initial state), Σ is a finite input alphabet, δ is a transition function from $Q \times \Sigma ~ \Rightarrow ~ Q$ and F is a set of final (accepting) states, a subset of Q. A string is accepted by such a device, if, starting from q_0, the sequence of transitions dictated by the tokens in the string ends up in one of the final states.

Table 2

center box tab(:); c | c s c | c c . :Input State:0:1 _ q0:q0:q1 q1:q1:q0

δ is usually specified as a table, which lists a new state for each state and input. As an example, a machine which accepts boolean strings of odd parity can be specified as $Q = \{ q_0, q_1 \}$, $\Sigma = \{ 0,1 \}$, $F = \{ q_1 \}$, and δ as shown in table 2.

Although such machines are usually described with fully explicit tables or graphs, transition functions can also be specified as a mathematical function of codes for the current state and the input. For example, variable-length parity can be specified as the exclusive-or of the current state and the input, each coded as a single bit. The primary result in the field of neural networks is that under simplified assumptions, networks have the capacity to perform arbitrary logical functions, and thus to act as finite-state controllers (McCulloch & Pitts, 1943; Minsky, 1972). In various configurations, modern multilayer feed-forward networks are also able to perform arbitrary boolean functions (Hornik et al., 1990; Lapedes & Farber, 1988; Lippman, 1987). Thus when used recurrently, these networks have the capacity to be any finite state recognizer as well. The states and tokens are assigned binary codes (say with one bit indicating which states are in F), and the code for the next state is simply computed by a set of boolean functions of the codes for current state and current input.

But the mathematical models for neural nets are "richer" than boolean functions, and more like polynomials. What does this mean for automata? In order not to confuse theory and implementation, I will first define a general mathematical object for language recognition as a forced discrete-time continuous-space dynamical system plus a precise initial condition and a decision function. The recurrent neural network architecture presented in the next section is a constrained implementation of this object.

By analogy to a finite-state recognizer, a *Dynamical Recognizer* is a quadruple $ "{" Z , SIGMA , OMEGA , G "}"$, where $Z ˜⊂˜ size 14 R sup size 12 k$ is a "space" of states and $z sub k (0)$ is the initial condition. $SIGMA$ is a finite input alphabet. $OMEGA$ is the "dynamic," a parameterized set (one for each token) of transformations on the space $omega sub {sigma sub i} ":" Z → Z$, and $G(Z) → "{" 0,1 "}"$ is the "decision" function.

Each finite length string of tokens in $SIGMA sup *$, $sigma sub 1, sigma sub 2, ... sigma sub n$, has a final state associated with it, computed by applying a precise sequence of transformations to the initial state: $z sub k (n)˜=˜ omega sub {sigma sub n} (... (omega sub {sigma sub 2} (omega sub {sigma sub 1} (z sub k (0)))))$. The language accepted and generated[1] by a dynamical recognizer is the set of strings in $SIGMA sup *$ whose final states pass the decision test.

In the "Mealy Machine" formulation (Mealy, 1955), which I use in the model below, the decision function applies to the penultimate state and the final token: $G(z

---

[1] To turn a recognizer into a generator, simply enumerate the strings in $SIGMA sup *$ and filter out those the recognizer rejects.

sub k (n-1) , sigma sub n) → "{" 0,1 "}"$. Just as in the case for finite automata, labeling the arcs rather than the nodes can often result in smaller machines.

There are many variants possible, but both $OMEGA$ and G must be constrained to avoid the vacuous case where some $omega$ or G is as powerful as a Turing Machine. For purposes of this paper, I will assume that G is as weak as a conventional neural network decision function, e.g. a hyperplane or a convex region, and that each $omega$ is as weak as a linear or quasi-linear transformation. G could also be a graded function instead of a forced decision, which would lead to a "more-or-less" notion of string acceptability, or it could be a function which returned a more complex categorization or even a *representation*, in which case I would be discussing dynamical *parsers*. Finally, one could generalize from discrete symbols to continuous symbols (MacLennan, 1989; Touretzky & Geva, 1987), or from discrete-time to continuous-time systems (Pearlmutter, 1989; Pineda, 1987).

There are some difficult questions which can be asked immediately about dynamical recognizers. What kind of languages can they recognize and generate? How does this mathematical description compare to various formal grammars on the grounds of parsimony, efficiency of parsing, neural and psychological plausibility, and learnability? I do not yet have the definitive answers to these questions, as this paper is the first study, but will touch on some of these issues later.

One thing is clear from the outset, that even a linear dynamical recognizer model can function as an arbitrary finite state automaton. The states of the automaton are "embedded" in a finite dimensional space such that a linear transformation can account for the state transitions associated with each token. Consider the case where each of k states is a k-dimensional binary unit vector (a 1-in-k code) Each $omega sub {sigma sub i}$ is simply a permutation matrix which "lists" the state transitions for each token, and the decision function is just a logical mask which selects those states in F. It is perhaps an interesting theoretical question to determine the minimum dimensionality of such a linear "embedding" for an arbitrary regular language.

With the introduction of non-linearities, more complex grammars can also be accounted for. Consider a one-dimensional system where Z is the unit line, $z sub 0 ~=~ 1$, G tests if $z(n) > .75$, and $SIGMA = "{" "L", "R" "}"$. If the transformation $omega sub "L"$ is "multiply z by 0.5" and $omega sub "R"$ is "multiply z by 2 modulo 2" (which only applies when z(i) is 0 or 1), then the recognizer accepts the balanced parentheses language. In other words, it is just as mathematically possible to embed an "infinite state machine" in a dynamical recognizer as it is to embed a finite state machine. I will return to these issues in the conclusion.

To begin to address the question of learnability, I now present and elaborate upon my earlier work on Cascaded Networks (Pollack, 1987a), which were used in a recurrent fashion to learn parity and depth-limited parenthesis balancing, and to map between word sequences and propositional representations (Pollack, 1990).

3. The Model

A Cascaded Network is a well-behaved higher-order (Sigma-Pi) connectionist architecture to which the back-propagation technique of weight adjustment (Rumelhart et al., 1986) can be applied. Basically, it consists of two subnetworks in a master-slave relationship: The *function* (slave) network is a standard feed-forward network, with or without hidden layers. However, the weights on the function network are dynamically computed by the linear *context* (master) network. A context network has as many outputs as there are weights in the function network. Thus the input to the context network is used to "multiplex" the function computed, a divide and conquer heuristic which can make learning easier.

Figure 1 near here

Figure 1. *A Sequential Cascaded Network. This is a recurrent version of a "higher order" or "sigma-pi" network in which the outputs of the master net (left)* **are** *the weights for the slave net (right), and the outputs of the slave net are recurrent inputs to the master net.*

When the outputs of the function network are used as recurrent inputs to the context network, a system can be built which learns to associate specific outputs for variable length input sequences. A block diagram of a Sequential Cascaded Network is shown in Figure 1. Because of the multiplicative connections, each input is, in effect, processed by a different function. Given an initial context, $z_k(0)$ (all .5's by default), and a sequence of inputs, $y_j(t),~ t=~1...n$, the network computes a sequence of output/state vectors, $z_i(t),~ t=~1...n$ by dynamically changing the set of weights, \$w sub ij (t)\$. Without hidden units, the forward-pass computation is:

which reduces to:

(1)

where \$g (v) = ˜ 1 / squash\$ is the usual sigmoid function used in back-propagation systems.

In previous work, I assumed that a teacher could supply a consistent and generalizable final-output for each member of a set of strings, which turned out to be a significant overconstraint. In learning a two-state machine like parity, this did not matter, as the 1-bit state fully determines the output. However, for the case of a higher-dimensional system, we may know what the final output of a system should be, but we *don't care* what its final state is.

Jordan (1986) showed how recurrent back-propagation networks could be trained with "don't care" conditions. If there is no specific target for an output unit during a particular training example, simply consider its error gradient to be 0. This will work, *as long as that same unit receives feedback from other examples.* When the don't-cares line up, the weights to those units will never change. One possible fix, so-called "Back Propagation through time" (Rumelhart et al., 1986), involves a complete unrolling of a recurrent loop and has had only modest success (Mozer, 1988), probably because of conflicts arising from equivalence constraints between interdependent layers. My fix involves a single *backspace*, unrolling the loop only once. For a particular string, this leads to the calculation of only one error term for each weight (and thus no conflict) as follows. After propagating the errors determined on only a subset of the weights from the "acceptance" unit:

The error on the remainder of the weights ($ {partial E} over {partial w sub ijk }~,~ i~!=~ a$) is calculated using values from the penultimate time step:

Figure 2 near here

Figure 2. *The "Backspace" Trick. Because only one desired output is supplied, only partial information is available for computing error gradients on the weights. The trick, which avoids completely unrolling the loop or carrying along information across all cycles, is to unroll the loop only once. The penultimate configuration of input/outputs is used to calculate gradients for the which are not directly connected to the known output unit.*

The schematic for this mode of back propagation is shown in figure 2, where the gradient calculations for the weights are highlighted. The method applies with small variations whether or not there are hidden units in the function or context network, and whether or not the system is trained with a single "accept" bit for desired output, or a larger pattern (representing a tree structure, for example (Pollack, 1990)). The important point is that the gradients connected to a subset of the outputs are calculated directly, but the gradients connected to don't-care recurrent states are calculated one step back in time. The forward and backward calculations are performed over a corpus of variable-length input patterns, and then all the weights are updated. As the overall squared sum of errors approaches 0, the network improves its calculation of final outputs for the set of strings in the training set. At some threshold, for example, when the network responds with above .8 for accept strings, and below .2 for reject strings, training is halted. The network now classifies the training set and can be tested on its generalization to a transfer set.

Unfortunately, for language work, the generalization must be infinite.

4. Induction as Phase Transition

In my original (1987) studies of learning the simple regular language of odd parity, I expected the network to merely implement "exclusive or" with a feedback link. It turns out that this is not quite enough. Because termination of back-propagation is usually defined as a 20% error (e.g. logical "1" is above 0.8), recurrent use of this logic tends to a limit point. In other words, separation of the finite exemplars is no guarantee that the network can recognize sequential parity in the limit. Nevertheless, this is indeed possible as illustrated by the figures below.

A small cascaded network composed of a 1-input 3-output function net (with bias connections, 6 weights for the context net to compute) and a 2-input 6-output context net (with bias connections, 18 weights) was trained on odd parity of a small set of strings up to length 5 (table 1). Of the 3 outputs, two were fed back recurrently as state, and the third was used as the accept unit. At each epoch, the weights in the network were saved in a file for subsequent study. After being trained for about 200 epochs, the network tested successfully on much longer strings. But it is important to show that the network is recognizing parity "in the limit".

Figure 3 near here

Figure 3. *Three stages in the adaptation of a network learning parity. (a) the test cases are separated, but there is a limit point for 1* at about 0.6. (b) after another epoch, the even and odd sequences are slightly separated. (c) after a little more training, the oscillating cycle is pronounced.*

In order to observe the limit behavior of a recognizer at various stages of adaptation, we can observe its response to either $SIGMA sup *$ or to a very long "characteristic string" (which has the best chance of breaking it). For parity, a good characteristic string is the sequence of 1's, which should cause the most state changes. Figure 3 shows three stages in the adaptation of a network for parity, by testing the response of three intermediate configurations to the first 100 strings of 1*. In the first figure, despite success at separating the small training set, a single attractor exists in the limit, so that long strings are indistinguishable. After another epoch of training, the even and odd strings are slightly separated, and after still further training, the separation is significant enough to drive a threshold through.

Figure 4 near here

Figure 4. *A bifurcation diagram showing the response of the parity-learner to the first 25 characteristic strings over 200 epochs of training. In each column representing an epoch, 25 points, corresonding to 25 different length strings of 1's, are plotted.*

This "phase transition" is shown more completely in figure 4. The vertical axis represents, again, the network's accept/reject response to characteristic strings, but the horizontal axis shows the evolution of this response across all 200 epochs. Each vertical column contains 25 (overlapping) dots marking the network's response to the first 25 characteristic strings. Thus, each "horizontal" line in the graph plots the evolution of the network's response to one of the 25 strings. Initially, all strings longer than length 1 are not distinguished. From epoch 60 to epoch 80, the network is improving at separating finite strings. At epoch 84, the network is still failing in the limit, but at epoch 85, the network undergoes a "bifurcation," where a small change in weights transforms the network's limit behavior from limit point to a limit cycle. This phase transition is so "adaptive" to the classification task that the network rapidly exploits it.

I want to stress that that this is a new and very interesting form of mechanical induction. Before the phase transition, the machine is in principle *not* capable of performing the serial parity task; after the phase transition it *is*, and this change in abilities is rapidly exploited by adaptive search. This kind of learning dynamic may be related to biological evolution through natural selection as well as to insight problem-solving (the "aha" phenomenon). The induction is not "one shot" or instantaneous, but more like a "punctuated equilibria" in evolution, where a "pre-adaptive" capacity enables a population some advantage which then drives very rapid change. Metcalfe & Wiebe (1987) report psychological experiments on insight problems in which human subjects measurably undergo a similar cognitive phase transition, reporting no progress on the problems until the solution appears.

---

[1] For the simple low dimensional dynamical systems usually studied, the "knob" or control parameter for such a bifurcation diagram is a scalar variable; here the control parameter is the entire 32-D vector of weights in the network, and back-propagation turns the knob.

5. Benchmarking Results

Connectionist and other machine learning algorithms are, unfortunately, very sensitive to the statistical properties of the set of exemplars which make up the learning environment or data-set. When researchers develop their own learning environments, there is a difficult methodological issue bearing on the status of repetitive data-set refinement, especially when experimental results bear on psychologically measured statistics, or the evolution of the data-set is considered too irrelevant to publish. This has correctly led some researchers to include the learning environment as a variable to manipulate (Plunkett & Marchman, 1989). Besides this complicated path, the other methodologically clean choices are to use "real world" noisy data, to choose data once and never refine it, or to use someone else’s published training data. For this paper, I chose to use someone else’s.

Table 3

center box; l | l . SET 1 ACCEPT SET 1 REJECT 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 = SET 2
ACCEPT SET 2 REJECT 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0
1 0 1 0 1 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 =
SET 3 ACCEPT SET 3 REJECT 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 0 1 0 0 0 1 1 1 0 1 0 0 1
0 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1
1 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 = SET 4
ACCEPT SET 4 REJECT 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 0 0 0 0 1 1 1 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 0 0 1 0 1 0 0 1 0 0 0
1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0

center box; l | l . SET 5 ACCEPT SET 5 REJECT 1 1 0 0 0 1 1 1 1 0 0 1 0 1 1 0 1 0 1 0
0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 0 0 0 1 1 1 1 0 1 0 1 0
1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 = SET 6
ACCEPT SET 6 REJECT 1 0 1 0 1 0 1 1 0 0 1 1 1 0 1 0 1 0 0 0 1 1 1 1 0 1 0 0 0 0 0 0 0 1
1 1 0 1 1 1 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 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 0 0 1 0 0 1 0 0 1 = SET 7 ACCEPT SET 7 REJECT 1 1
0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 0
0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 1 1 0 1 0 0 1 0 1 0 1 1 0 0 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

Tomita (1982) performed elegant experiments in inducing finite automata from positive and negative exemplars. He used a genetically inspired two-step hill-climbing procedure, which manipulated 9-state automata by randomly adding, deleting or moving transitions, or inverting the acceptability of a state. Starting with a random machine, the current machine was compared to a mutated machine, and changed only when an improvement was made in the result of a heuristic evaluation function. The first hill-climber used an evaluation function which maximized the difference between the number of positive examples accepted and the number of negative examples accepted. The second hill-climber used an evaluation function which maintained correctness of the examples while minimizing the automaton's description (number of states, then number of transitions) Tomita did not randomly choose his test cases, but instead, chose them consistently with seven regular languages he had in mind (see table 3). The difficulty of these problems lies not in the languages Tomita had in mind, but in the arbitrary and impoverished data sets he used.

Table 4

center tab(:); c c l c . Language #:Description _ 1:1* 2:(1 0)* 3:no odd zero strings after odd 1 strings 4:no 000's 5:pairwise, an even sum of 01's and 10's. 6:number of 1's - number of 0's = 0 mod 3 7:0*1*0*1*

Each training environment was simply defined by two sets of boolean strings, which are given in Table 4. For uniformity, I ran all seven cases, as given, on a sequential cascaded network of a 1-input 4-output function network (with bias connections, making 8 weights for the context net to compute) and a 3-input 8-output context network with bias connections. The total of 32 context weights are essentially arranged as a 4 by 2 by 4 array. Only three of the outputs of the function net were fed back to the context network, while the fourth output unit was used as the accept bit. The standard back-propagation learning rate was set to 0.3 and the momentum to 0.7. All 32 weights were reset to random numbers between ±0.5 for each run. Training was halted when all accept strings returned output bits above 0.8 and reject strings below 0.2.

5.1. Results

Table 5

center box; c | c | c | c l | n | n | n . No. Mutations Avg. Epochs % Convergent Language (Hill-Climber) (Backprop) (Backprop) _ 1 98 54 100 2 134 787 20 3 2052 213 70 4 442 251 100 5 1768 637 80 6 277 0 7 206 595 50

Of Tomita's 7 cases, all but data-sets #2 and #6 converged without a problem in several hundred epochs. Case 2 would not converge, and kept treating negative case 110101010 as correct; I had to modify the training set (by adding reject strings 110 and 11010) in order to overcome this problem. Case 6 took several restarts and thousands of cycles to converge.

In the spirit of the machine learning community, I recently ran a series of experiments to make these results more empirical. Table 5 compares Tomita's stage one "number of mutations" to my "average number of epochs". Because back-propagation is sensitive to initial conditions (Kolen & Pollack, 1990), running each problem once does not give a good indication of its difficulty, and running it many times from different random starting weights can result in widely disparate timings. So I ran each problem 10 times, up to 1000 epochs, and averaged only those runs which separated the training sets (accepts above .6; rejects below .4). The column labeled "% Convergent" shows the percent of the 10 runs for each problem which separated the accept and reject strings within 1000 cycles. Although it is difficult to compare results between completely different methods, taken together, the averaged epochs and the percent convergent numbers give a good idea of the difficulty of the Tomita data-sets for my learning architecture.

5.2. Analysis

Figure 5 near here

Figure 5. *The minimal FSA's recognizing Tomita's 7 data sets. A finite machine consists of a set of states, shown as circles, an initial state, indicated by the lone arrow, and a subset of states called final states, which are indicated by double circles. The transitions from state to state are arrows labeled with tokens from the vocubulary (0 or 1).*

Figure 6 near here

Figure 6. *Because it is not clear what languages are being generated by the 7 trained networks, these figures show the training set (with 'X's) and parts of the languages induced (only for strings of up to 9 bits long). Each figure contains 9 rows of rectangles, which are colored white for reject and black for accept. The first row is the response to the strings "0" and "1", the second row is the response to the strings "00," "01," "10," and "11," and the third is for "000," "001," and so on. It is engineered so one could locate a network's respons to a particular string, by moving one's eyes down through a figure.*

Tomita ran a brute-force enumeration to find the minimal automaton for each language, and verified that his hill-climber was able to find them. This are displayed in Figure 5. Unfortunately, I ran into some difficulty trying to figure out exactly which finite state automata (and regular languages) were being induced by my architecture.

For this reason, in figure 6, I present "prefixes" of the languages recognized and generated by the seven first-run networks. Each rectangle assigns a number, 0 (white) or 1 (black), to all boolean strings up to length 9 (which is the limit of visibility), and thus indicates, in black, the strings which are accepted by the respective network. Starting at the top of each rectangle, each row r contains $2 sup r$ subrectangles for all the strings of length r in lexical order, so the subrectangle for each string is sitting right below its prefix. The top left subrectangle shows a number for the string 0, and the top right shows a number for the string 1. Below the subrectangle for 0 are the subrectangles for the strings 00 and 01, and so on. The training sets (Table 4) are also indicated in these figures, as inverted "X's" in the subrectangles corresponding to the training strings.

Note that although the figures display some simple recursive patterns, none of the ideal minimal automata were induced by the architecture. Even for the first language 1*, a 0 followed by a long string of 1's would be accepted by the network. My architecture generally has the problem of not inducing "trap" or error states. It can be argued that other FSA inducing methods get around this problem by presupposing rather than learning the trap states.[2]

Figure 7 near here

Figure 7. *The number of states in the 7th machine grew dramatically as $epsilon$ was lowered.*

If the network is not inducing the smallest consistent FSA, what is it doing? The physical constraint that an implemented network use *finitely specified* weights means that the states and their transitions cannot be arbitrary -- there must be some geometric

[2] Tomita assumed a trap state which did not mutate, and (Servan-Schreiber et al., 1989) compared the incoming token to a thresholded set of token predictions, trapping if the token was not predicted.

relationship among them.

Based upon the studies of parity, my initial hypothesis was that a set of clusters would be found, organized in some geometric fashion: i.e. an embedding of a finite state machine into a finite dimensional geometry such that each token's transitions would correspond to a simple transformation of space. I wrote a program which examined the state space of these networks by recursively taking each unexplored state and combining it with both 0 and 1 inputs. A state here is a 3-dimensional vector, values of the three recurrently used output units. To remove floating-point noise, the program used a parameter $epsilon$ and only counted states in each $epsilon$-cube once. Unfortunately, some of the machines seemed to grow drastically in size as $epsilon$ was lowered. In particular, Figure 7 shows the log-log graph of the number of unique states versus $epsilon$ for the machine resulting from training environment 7. Using the method of (Grassberger & Procaccia, 1983) this set was found to have a correlation dimension of 1.4 - good evidence that it is "fractal".

Figure 8 near here

Figure 8. *Images of the state spaces for the 7 training environments. Each network had 3 output units which could vary between 0 and 1, and so a "state" of the network can be visualized as a point in a 3-D cube. Each figure displays 2048 such states, corresponding to all boolean strings up to length 10.*

Because the states of the benchmark networks are "in a box" (Anderson et al., 1977) of low dimension, we can view these machines graphically to gain some understanding of how the state space is being arranged. Each 3-d state vector is plotted as a point in the unit cube. Partial graphs of the state spaces for the first-run networks are shown in Figure 8. States were computed for all boolean strings up to and including length 10, so each figure contains 2048 points, often overlapping.

The images (a) and (d) are what I initially expected, clumps of points which closely map to states of equivalent FSA's. Images (b) and (e) have limit "ravines" which can each be considered states as well. However, the state spaces, (c), (f), and (g) of the dynamical recognizers for Tomita cases 3, 6, and 7, are interesting, because, theoretically, they are *infinite* state machines, where the states are not arbitrary or random, requiring an infinite table of transitions, but are constrained in a powerful way by mathematical principle.

In thinking about such a principle, consider systems in which extreme observed complexity emerges from algorithmic simplicity plus computational power. When I first saw some of the state space graphs (Figure 8), they reminded me of Barnsley's Iterated Functional Systems (Barnsley, 1988), where a compactly coded set of affine transformations is used to *iteratively* construct displays of fractals, previously described *recursively* using line-segment rewrite rules (Mandelbrot, 1982). The calculation is simply the repetitive transformation (and plotting) of a state vector by a sequence of randomly chosen affine transformations. In the infinite limit of this process, fractal "attractors" emerge (e.g. the widely reproduced fern).[3]

By eliminating the sigmoid, commuting the y_j and z_k terms in Eq. 1:

and treating the y_j's as an infinite random sequence of binary unit vectors (1-in-j codes), the forward pass calculation for my network can be seen as the same process used in an Iterated Function System (IFS). Thus, my figures of state-spaces, which emerge from the projection of Σ^* into Z, are fractal attractors, as defined by Barnsley.

6. Related Work

The architecture and learning paradigm I used is also being studied by Lee Giles and colleagues, and is closely related to the work of Elman and Servan-Schreiber et al on Simple Recurrent Networks. Both architectures rely on extending Jordan's recurrent networks in a direction which separates visible output states from hidden recurrent states, without making the unstable "back-propagation through time" assumption. Besides our choice of language data to model, the two main differences are that:

[3] Barnsley's use of the term "attractor" is different than the conventional use of the term given in the introduction, yet is technically correct in that it refers to the "limit" of a iterative process. It can be thought of as what happens when you randomly drop an infinite number of microscopic iron filing "points" onto a piece of paper with a magnetic field lying under it; each point will land "on" the underlying attractor.

(1) They use a "predictive" paradigm, where error feedback is provided at every time step in the computation, and I used a "classification" paradigm, feeding back only at the end of the given examples. Certainly, the predictive paradigm is more psychologically plausible as a model of positive only presentation (c.f., Culicover & Wexler, pp 63-65), but the Tomita learning environments are much more impoverished.

I have no commitment to negative information; all that is required is some desired output which discriminates among the input strings in a generalizable way. Positive versus negative evidence is merely the simplest way (with 1 bit) to provide this discrimination.

(2) They use a single layer (first order) recurrence between states, whereas I use a higher order (quadratic) recurrence. The multiplicative connections are what enable my model to have "fractal" dynamics equivalent in the limit to an IFS, and it may be that the first-order recurrence, besides being too weak for general boolean functions (Minsky & Papert, 1988) and thus for arbitrary regular languages (such as parity), also results only in simple steady-state or periodic dynamics.

Besides continued analysis, scaling the network up beyond binary symbol alphabets and beyond syntax, immediate follow-up work will involve comparing and contrasting our respective models with the other two possible models, a higher order network trained on prediction, and a simple recurrent network model trained on classification.

7. Conclusion

Figure 9 near here

Figure 9. *A one-dimensional dynamical recognizer for balanced parentheses, which is a context-free language. If an "L" is received, the state is divided by 2, if an "R" is received, the state is multiplied by 2 (modulo 2). This induces a "fractal" state space on the unit line which can be cut at .75*

to determine if a string is grammatical.

If we take a state space picture of the one-dimensional dynamical recognizer for parenthesis balancing developed earlier, it looks like Figure 9. An infinite state machine is embedded in a finite geometry using "fractal" self-similarity, and the decision function is cutting through this set. The emergence of these fractal attractors is interesting because I believe it bears on the question of how neural-like systems could achieve the power to handle more than regular languages.

Because it is quite clear that human languages are not formal, connectionists can maintain that recursive computational power is not of the "essence of human computation."[4] However, it is also quite clear that without understanding the issue of generative capacity, connectionists and others working on naturalistic computation, can stumble again and again into the trap of making strong claims for their models, easy to attack for not offering an adequate replacement for established theory. (Fodor & Pylyshyn, 1988; Pinker & Prince, 1988). But it is only because of "long-term lack of competition" that descriptive theories involving rules and representations can be defended as explanatory theories. Here is an alternative hypothesis for complex syntactic structure:

> *The state-space limit of a dynamical recognizer, as $SIGMA sup * -> SIGMA sup inf$, is an Attractor, which is cut by a threshold (or similar decision) function. The complexity of the generated language is regular if the cut falls between disjoint limit points or cycles, context-free if it cuts a "self-similar" (recursive) region, and context-sensitive if it cuts a "chaotic" (pseudo-random) region.*

There is certainly substantial need for work on the theoretical front to more thoroughly formalize and prove or disprove the six main theorems implied by my hypothesis. I do not expect the full range of context-free or context sensitive systems to be covered by conventional quasi-linear processing constraints, and the question remains wide open as to whether the syntactic systems which can be described by neural dynamical recognizers have any convergence with the needs of natural language systems.

---

4 (Rumelhart & McClelland, 1986), p. 119.

Because information processing provides the "essence" of complex forms of cognition, like language, it is important to understand the relationship between complex emergent behaviors of dynamical systems (including neural systems) and traditional notions of computational complexity, including the Chomsky hierarchy as well as algorithmic information theory (Chaitin, 1966), but the study of this relationship is still in its infancy.

In (Pollack, 1987b), I constructed a Turing Machine out of connectionist parts, and essentially showed that rational values, constants, precise thresholds, and multiplicative connections (all used in the sequential cascaded network architecture) were sufficient primitives for computationally universal recurrent neural networks.

Cellular Automata, which we might view as a kind of low-density, synchronous, uniform, digital restriction of neural networks, have been studied as dynamical systems (Wolfram, 1984) and proven to be as powerful as universal Turing Machines, e.g. (Lindgren & Nordahl, 1990). Furthermore, (Moore, 1990) has shown that there are simple mathematical models for dynamical systems which are also universal, and it follows directly that determination of the behavior of such dynamical systems in the limit is undecidable and unpredictable, even with precise initial conditions. In stronger terms, the theoretical foundations of computer and information science may be in accord with the lack of predictability in the universe.

Finally, Crutchfield and Young (1989) have studied the computational complexity of dynamical systems reaching the onset of chaos via period-doubling. They have shown that these systems are not regular, but are finitely described by Indexed Context-Free Grammars. It may, of course, be just be a coincidence that several modern computational linguistic grammatical theories also fall in this class (Joshi, 1985; Joshi et al., 1989; Pollard, 1984).

In conclusion, I have merely illuminated the possibility of the existence of a naturalistic alternative to explicit recursive rules as a description for the complexity of language. Such a mathematical description would be compact, "parsimonious" in fact, since the infinite state machine does not require infinite description, but only a finitely described set of weights. It was shown to be feasible to learn this type of description from a finite set of examples using pseudo-continuous hill-climbing parameter-adaptation (in other words, back-propagation). However, performance in the limit appears to jump in discrete steps, inductive phase transitions which might correspond to psychological "stages" of acquisition. Finally, the languages so described can be recognized and generated efficiently by neural computation systems.

8. Acknowledgments

This work has been partially sponsored by the Office of Naval Research under grant N00014-89-J-1200. Thanks to the numerous colleagues who have discussed and/or criticized various aspects of this work or my presentation of it, including: T. Bylander, B. Chandrasekaran, J. Crutchfield, L. Giles, E. Gurari, S. Hanson, R. Kasper, J. Kolen, W. Ogden, T. Patten, R. Port, K. Supowit, P. Smolensky, D. S. Touretzky, and A. Zwicky.

9. References

Anderson, J. A., Silverstein, J. W., Ritz, S. A. & Jones, R. S. (1977). Distinctive Features, Categorical Perception, and Probability Learning: Some Applications of a Neural Model. *Psychological Review*, *84*, 413-451.

Angluin, D. (1978). On the complexity of minimum inference of regular sets. *Information and Control*, *39*, 337-350.

Angluin, D. & Smith, C. H. (1983). Inductive Inference: Theory and Methods. *Computing Surveys*, *15*, 237-269.

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

Berwick, R. (1985). *The Acquisition of Syntactic Knowledge*. Cambridge: MIT Press.

Chaitin, G. J. (1966). On the length of programs for computing finite binary sequences. *Journal of the AM, 13*, 547-569.

Chomsky, N. (1956). Three Models for the Description of Language. *IRE Transactions on Information Theory, IT-2*, 113-124.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press.

Crutchfield, J. P., Farmer, J. D., Packard, N, H. & Shaw, R. S. (1986). Chaos. *Scientific American*, *255*, 46-57.

Crutchfield, J. P & Young, K. (1989). Computation at the Onset of Chaos. In W. Zurek, (Ed.), *Complexity, Entropy and the Physics of INformation*. Reading, MA: Addison-Wesley.

Derrida, B. & Meir, R. (1988). Chaotic behavior of a layered neural network. *Phys. Rev. A*, *38*.

Devaney, R. L. (1987). *An introduction to chaotic dynamical systems*. Reading, MA: Addison-Wesley.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science, 14*, 179-212.

Feldman, J. A. (1972). Some Decidability Results in grammatical Inference. *Information & Control, 20*, 244-462.

Fodor, J. & Pylyshyn, A. (1988). Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition, 28*, 3-71.

Giles, C. L., Sun, G. Z., Chen, H. H., Lee, Y. C. & Chen, D. (1990). Higher Order Recurrent Networks and Grammatical Inference. In D. S. Touretzky, (Ed.), *Advances in Neural Information Processing Systems*. Los Gatos, CA: Morgan Kaufman.

Gleick, J. (1987). *Chaos: Making a new science*. New York: Viking.

Gold, E. M. (1967). Language Identification in the Limit. *Information & Control, 10*, 447-474.

Gold, E. M. (1978). Complexity of automaton identification from given data. *Information and Control, 37*, 302-320.

Grassberger, P. & Procaccia, I. (1983). Measuring the Strangeness of Strange Attractors. *Physica, 9D*, 189-208.

Grebogi, C., Ott, E. & Yorke, J. A. (1987). Chaos, Strange Attractors, and Fractal Basin Boundaries in Nonlinear Dynamics. *Science, 238*, 632-638.

Hendin, O., Horn, D. & Usher, M. (1991). Chaotic Behavior of a neural network with dynamical thresholds. *Int. Journal of Neural Systems, To appear*.

Hopfield, J. J. (1982). Neural Networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, *79*, 2554-2558.

Hornik, K., Stinchcombe, M. & White, H. (1990). Multi-layer Feedforward Networks are Universal Approximators. *Neural Networks, 3,* 551-560.

Huberman, B. A. & Hogg, T. (1987). Phase Transitions in Artificial Intelligence Systems. *Artificial Intelligence, 33*, 155-172.

Joshi, A. K. (1985). Tree Adjoining Grammars: How much context-sensitivity is required to provide reasonable structural descriptions?. In D. R. Dowty, L. Karttunen & A. M. Zwicky, (Eds.), *Natural Language Parsing*. Cambridge: Cambridge University Press.

Joshi, A. K., Vijay-shanker, K. & Weir, D. J. (1989). Convergence of mildly context-sensitive grammar formalism.. In T. Wasow & P. Sells, (Eds.), *The Processing of Linguistic Structure*. Cambridge: MIT Press.

Kolen, J. F. & Pollack, J. B. (1990). Backpropagation is Sensitive to Initial Conditions. *Complex Systems, 4*, 269-280.

Kurten, K. E. (1987). Phase transitions in quasirandom neural networks. In *Institute of Electrical and Electronics Engineers First International Conference on Neural Networks*. San Diego, II-197-20.

Lapedes, A. S. & Farber, R. M. (1988). How Neural Nets Work. LAUR-88-418: Los Alamos.

Lieberman, P. (1984). *The Biology and Evolution of Language*. Cambridge: Harvard University Press.

Lindgren, K. & Nordahl, M. G. (1990). Universal Computation in simple one-dimensional cellular automata. *Complex Systems, 4*, 299-318.

Lippman, R. P. (1987). An introduction to computing with neural networks. *Institute of Electrical and Electronics Engineers ASSP Magazine, April*, 4-22.

MacLennan, B. J. (1989). Continous Computation. CS-89-83, Knoxville: Computer Science Dept, University of Tennessee.

MacWhinney, B. (1987). *Mechanisms of Language Acquisition*. Hillsdale: Lawrence Erlbaum Associates.

Mandelbrot, B. (1982). *The Fractal Geometry of Nature*. San Francisco: Freeman.

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115-133.

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal, 43*, 1045-1079.

Metcalfe, J. & Wiebe, D. (1987). Intuition in insight and noninsight problem solving. *Memory and Cognition, 15*, 238-246.

Minsky, M. (1972). *Computation: Finite and Infinite Machines*. Cambridge, MA: MIT Press.

Minsky, M. & Papert, S. (1988). *Perceptrons*. Cambridge, MA: MIT Press.

Moore, C. (1990). Unpredictability and undecidability in dynamical systems. *Physical Review Letters, 62*, 2354-2357.

Mozer, M. (1988). A focused Back-propagation Algorithm for Temporal Pattern Recognition. CRG-Technical Report-88-3: University of Toronto.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation, 1*, 263-269.

Pineda, F. J. (1987). Generalization of Back-Propagation to Recurrent Neural Networks. *Physical Review Letters, 59*, 2229-2232.

Pinker, S. (1984). *Language Learnability and Language Development*. Cambridge: Harvard University Press.

Pinker, S. & Prince, A. (1988). On Language and Connectionism: Analysis of a parallel distributed processing model of language inquisition.. *Cognition, 28*, 73-193.

Pinker, S. & Bloom, P. (1990). Natural Language and Natural Selection. *Brain and Behavioral Sciences, 12*, 707-784.

Plunkett, K. & Marchman, V. (1989). Pattern Association in a Back-propagation Network: Implications for Child Language Acquisition. Technical Report 8902, San Diego: UCSD Center for Research in Languge.

Pollack, J. B. (1987). Cascaded Back Propagation on Dynamic Connectionist Networks. In *Proceedings of the Ninth Conference of the Cognitive Science Society*. Seattle, 391-404.

Pollack, J. B. (1987). On Connectionist Models of Natural Language Processing. Ph.D. Thesis, Urbana: Computer Science Department, University of Illinois. (Available as MCCS-87-100, Computing Research Laboratory, Las Cruces, NM)

Pollack, J. B. (1989). Implications of Recursive Distributed Representations. In D. S. Touretzky, (Ed.), *Advances in Neural Information Processing Systems*. Los

Gatos, CA: Morgan Kaufman.

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

Pollard, C. (1984). Generalized Context-Free Grammars, Head Grammars and Natural Language.. Doctoral Dissertation, Palo Alto: Dept of Linguistics, Stanford University.

Rivest, R. L. & Schapire, R. E. (1987). A new approach to unsupervised learning in deterministic environments. In *Proceedings of the Fourth International Workshop on Machine Learning*. Irvine, 364-475.

Rumelhart, D. E., Hinton, G. & Williams, R. (1986). Learning Internal Representations through Error Propagation. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 1. Cambridge: MIT Press.

Rumelhart, D. E. & McClelland, J. L. (1986). PDP Models and General Issues in Cognitive Science. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 1. Cambridge: MIT Press.

Servan-Schreiber, D., Cleeremans, A. & McClelland, J. L (1989). Encoding Sequential Structure in Simple Recurrent Networks. In D. S. Touretzky, (Ed.), *Advances in Neural Information Processing Systems*. Los Gatos, CA: Morgan Kaufman.

Skarda, C. A. & Freeman, W. J. (1987). How brains make chaos. *Brain & Behavioral Science, 10*.

Smolensky, P. (1986). Information Processing in Dynamical Systems: Foundations of Harmony Theory. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 1. Cambridge: MIT Press.

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

Touretzky, D. S. & Geva, S. (1987). A distributed connectionist representation for concept structures. In *Proceedings of the Ninth Annual Conference of the*

Cognitive Science Society. Seattle, 155-164.

van der Maas, H., Verschure, P. & Molenaar, P. (1990). A note on chaotic behavior in simple neural networks. In *Neural Networks*. 3, 119-122.

Wexler, K. & Culicover, P. W. (1980). *Formal Principles of Language Acquisition*. Cambridge: MIT Press.

Wolfram, S. (1984). Universality and Complexity in Cellular Automata. *Physica, 10D*, 1-35.

TABLE 1 - What is the rule which defines the language?

TABLE 2 - The transition table for a two state machine which defines the regular language of odd parity (binary strings with an odd number of 1's). Starting in state q0, each 1 received will move the state back and forth between q0 and q1, and will accept strings which end in state q1.

TABLE 3 - Training data for seven languages from Tomita (1982).

TABLE 4 - Brief descriptions of the seven regular languages corresponding to the training data from Tomita (1982). The '*' indicates an expression to be repeated one or more times. Minimal FSAs which generate these languages are also shown in figure 5.

TABLE 5 Performance comparison between Tomita's (Hill Climber) and Pollack's (Backprop) model. Because back-propagation doesn't always converge, 10 trials with different initial weights were run, and those which did not separate the training set within 1000 epochs could not be counted. The ease of the problem for the architecture is thus also measured by the column labeled "% convergent."