Fractal Encoding of Context Free Grammars in Connectionist Networks

Whitney Tabor
University of Connecticut

(2000) Expert Systems 17(1), pp. 41-56

Please address correspondence to:

Whitney Tabor
Department of Psychology
University of Connecticut
Storrs, CT 06269 USA
(860) 486-4910 (Phone)
(860) 486-2760 (Fax)
tabor@uconn.edu

Abstract: Connectionist network learning of context free languages has so far been applied only to very simple cases and has often made use of an external stack. Learning complex context free languages with a homogeneous neural mechanism looks like a much harder problem. The current paper takes a step toward solving this problem by analyzing context free grammar computation (without addressing learning) in a class of analog computers called *Dynamical Automata*, which are naturally implemented in connectionist networks. The result is a widely applicable method of using fractal sets to organize infinite state computations in a bounded state space. An appealing consquence is the development of parameter-space maps, which locate various complex computers in spatial relationships to one another. An example suggests that such a global perspective on the organization of the parameter space may be helpful for solving the hard problem of getting connectionist networks to learn complex grammars from examples.

1. Introduction

Smolensky (1990) argues that connectionist (or "neural") networks offer an opportunity to overcome the brittleness of symbolic devices without foregoing their powerful computational capabilities. "Brittleness" refers to the fact that many symbolic devices are catastrophically sensitive to small distortions in their encoding---a bit or a semicolon out of place can bring an entire system to its knees. Such sensitivity is reminiscent of the trademark behavior of "chaotic" dynamical processes: small differences in initial conditions give rise to substantial differences in long-term behavior. It would be ironic, then, if the interpretation of connectionist devices as dynamical systems with potentially chaotic behaviors led to a realization of Smolensky's ideal. Intriguingly, this is the character of a number of recent results connecting symbolic computation with multi-stable dynamical systems (Barnsley, 1993[1988]; Pollack, 1991; Moore, 1998; Blair and Pollack, in press; Tino and Dorffner, 1998; Tino, 1999). Fractal objects, which turn up as the traces of chaotic processes, turn out to be especially useful for instantiating powerful computing devices in metric space computers which exhibit graceful modification under small distortions. It is as though by embracing the caprice of a chaotic process, a computational system can stay in its good graces and make effective use of its complexity (cf. Crutchfield and Young, 1990; Crutchfield, 1994). Previous work has focused on how to instantiate complex symbolic computers in metric space computers like connectionist networks. The current work suggests that the most useful contribution of the metric space perspective is the revelation of geometric relationships among familiar, effective symbolic devices. It becomes possible to see, from a global, topological perspective how one symbolic computer can be gradually transformed into another one.

Recently, fractals have been used to organize computation in several dynamical systems for symbol processing (Jeffrey, 1990). Tino (1999) has shown that when points on such fractals are associated with probabilities, then fractal dimension is, in a useful sense, the geometric analog of entropy. Tino and Dorffner (1998) have applied the results to connectionist networks learning unknown time series and found that using the fractal structure to set the recurrent weights leads to an improvement over Simple Recurrent Network (SRN; Elman, 1991) and variable memory Markov model approaches (e.g., Ron, Singer, and Tishby, 1996). The current work complements these results by analyzing connectionist representations of specific infinite-state languages.

A number of researchers have studied the induction of context free grammars (CFGs) by connectionist networks. Many have used an external stack (Giles, Sun, Chen, Lee, and Chen; 1990; Sun, Chen, Giles, Lee, and Chen, 1990; Das, Giles, and Sun, 1993; Mozer and Das, 1993; Zheng, Goodman, and Smyth, 1994). Some have used more standard architectures (Wiles and Elman, 1995; Rodriguez, Wiles, and Elman, 1999). In all cases, only very simple context free languages have been learned. It is desirable to be able to handle more complex languages. It is also desirable to avoid an external stack: such a stack makes it harder to see the relationship between CFG nets and other more homogeneous connectionist architectures and it absolves the network from responsibility for the challenging part of the task---keeping track of an unbounded memory---thus making its

accomplishment fairly similar to another well-studied case, the learning of finite state languages (e.g., Servan-Schreiber, Cleeremans, and McClelland, 1991; Zheng, 1994; Casey, 1996).

This paper takes a step toward addressing these shortcomings by providing a representational analysis of neurally inspired devices called *Dynamical Automata* or DAs which can recognize all context free languages as well as many other languages. The approach is less ambitious than the models just cited in that learning is not attempted. On the other hand, it is more ambitious in that a representational analysis of the structural principles governing the DAs', and corresponding networks', computations is formally worked out for all context free languages. The essential principle, consistent with the experiments of Pollack (1991) the analysis of Moore (1998) is that fractal sets provide a method of organizing recursive computations in a bounded state space. The networks are recurrent, use linear and threshold activation functions and gating units, and may have stochastic units, but have no external stack. The representations used in DAs resemble the representations developed by Elman's Simple Recurrent Network when it is trained on syntax tasks (Elman, 1991) and which has been widely used in cognitive modeling, thus suggesting a set of principles that may be useful in building cognitively plausible neural models of syntax.

1.1. Overview

In Section 2 of this paper, I review previous studies that have used fractal sets to organize complex computation by connectionist devices.

Section 3 investigates a subclass of Dynamical Automata called Pushdown Dynamical Automata (PDDAs). These DAs emulate Pushdown Automata (PDAs) and are closely related to a type of "Dynamical Recognizer" that Moore (1998) proposed for recognizing Context Free Languages (CFLs). While Moore's case is one-dimensional, I describe a higher dimensional species which is naturally implemented in high-dimensional connectionist networks. Section 2 also contains a lemma that helps one choose appropriate initial conditions for Dynamical Automata (and Dynamical Recognizers) that process context-free languages.

Section 4 makes explicit how to go about encoding PDDAs and other Dynamical Automata in connectionist networks.

Section 5 focuses on the appealing consequence of performing complex computations in a metric space that I noted above: the metric provides a way of mapping out spatial relationships between machines with different computational properties. The examples of Section 5 suggest that such maps may be useful in addressing the difficult problem of learning complex grammars from data.

2. Examples of Fractal Computers

2.1. Example: Pollack (1991)

Pollack (1991) noted that a very simple artificial neural device could recognize the Dyck Language---the language in which left parentheses always precede corresponding right parentheses. He describes a machine along the lines of that shown in Figure 1.

----------Insert Figure 1 about here----------

Initially, the activation of unit z is 1. If a left parenthesis is presented, the network activates unit L which has the effect of allowing transmission of activation along the connection labeled $w_L = 1/2$. Similarly, if a right parenthesis is presented, the network activates unit R which allows transmission of activation along the connection labeled $w_R = 2$. With each presentation of a symbol, z updates according to the rule $z(t+1) = f\left(\sum_i w_i a_i \right) = f(w_L \circ z(t) + w_R \circ z(t)) =$ either $f(w_L \circ z(t))$ or $f(w_R \circ z(t))$. The activation function f(x) is equal to x for $x \in (0, 2]$ and equal to 2 for $x > 2$. Unit P is a threshold unit that becomes active if $z > 0.75$. Unit Q is a self-reinforcing threshold unit that is initially inactive but becomes active and stays active if z ever exceeds 1.5. Unit A is a threshold unit that computes P AND ~Q. Note that unit A becomes activated at the end of any string in which right parentheses follow and match left parentheses.

During the processing of grammatical strings, the activations of the z unit lie on the geometric series fractal, $\{1/2^n: n \in N\}$. In essence, this unit is a counter that keeps track of how many right parentheses are required to complete the string at any point. Although one could also use the set of non-negative integers, N, to perform the same function, the use of the bounded fractal permits the connectionist device to work with units of bounded activation. This simple example thus provides an indication of how fractal objects are useful in forming neural recognizers for infinite-state languages.

2.2. Example: Rodriguez, Wiles, and Elman (1999)

Wiles and Elman (1995) study a backpropagation network that is trained on samples from the closely-related language, $l^n r^n$, $n \in \{1,2,3...\}$. The model is presented with a sequence from $(l^n r^n)^*$ where n is randomly chosen from {1,..., 11} at each iteration. The task of the model is to predict successor symbols at each point. After many training episodes with different initial weight settings, Wiles and Elman found one network that generalized the pattern up to n = 18 (i.e. it performed as though it were recognizing $l^n r^n$ for $n \in \{1,...,18\}$).

Rodriguez et al. (1999) noted that networks like Wiles and Elman's can be viewed as nonlinear dynamical systems. They analyzed the corresponding linear systems which closely approximated the behavior of the nonlinear systems and found that the computation of $l^n r^n$ was organized around a saddle point in two dimensions:[1] when the network was

[1] See Perko (1991). A visualizable example is a system whose states consists of the positions of a water drop on mountainous terrain. Gravity pulls the water drop downhill at all times. A saddlepoint in this system is a

receiving a string of l's, it was iterating the map associated with the stable manifold of the saddle point---in effect it was computing successive values of $x(t) = x_0 e^{-kt}$ for x_0 the initial state, some positive k, and t = 0, 1, 2, 3,.... When it was receiving the corresponding string of r's it was iterating the map associated with the unstable manifold (same situation except $k < 0$ and the points are spread out along a different axis). With equally spaced values of t, the exponential equation $x(t) = x_0 e^{-kt}$ generates points on a geometric series fractal. Thus again, a parenthesis balancer is using geometric series fractals for its computation, this time along two different dimensions (the distinction between dimensions is a handy way of distinguishing the l and r states).

These two examples have shown how a particular type of fractal is useful for modeling parenthesis-balancing languages. This is helpful, but it is a very simple case. The next two examples describe more complex cases.

2.3. Example: Moore (1998)

Moore (1998) proposes Dynamical Recognizers, which compute on a metric space, X, by invoking a distinct map f_i: $X \rightarrow X$ for each symbol i. The Recognizer must be at a specified point in X when it starts reading a string. If it lands in a specified final region after processing each symbol of the string, then the string is deemed part of its language.

Moore uses Cantor sets (e.g., Strogatz, 1994) to construct recognizers on the one-dimensional metric space, $\Re$, for any context free language:

----------Insert Table 1 about here----------

Here, $1 \leq i \leq m$ and $0 < \alpha \leq 1/(3m + 1)$, where m is the number of symbols in the stack alphabet. The initial state (x_0) is 1/2. The final region is (0, 1). These 2m functions can be used to simulate a pushdown stack with m symbols. The functions "$push_i$" and "pop_i" correspond to "push symbol i onto the stack" and "pop symbol i off of the stack" respectively. It can be shown that pushdown automata generate the same languages as the subset of them in which there are no control state changes (Hopcroft and Ullman, 1979). Thus all context free languages can be generated by a device that consists simply of a pushdown stack and its control rules. Therefore, by composing $push_i$'s and pop_i's appropriately, Moore's functions can be used to build a mechanism for generating any context-free language.

There are two technical details that need to be looked after. First, how can Moore's recognizer detect an illegal pop, i.e., a move of the form $pop_j \circ push_i$ where $i \neq j$? Second, what guarantees that each sequence of pushes and pops is uniquely associated with a point in $\Re$?

---

pass separating two peaks and two valleys. The stable manifold of such a saddlepoint is the ridgeline between the two peaks.

Regarding the first question (illegal pops), Moore's mechanism is cleverly designed to allow easy detection of illegal pops: any move or partial move of the illegal form $pop_j \circ push_i$ makes $|x| \geq 2$ but all legal moves keep $|x| \leq 1$. In order to allow his device to record the information about whether $|x|$ ever exceeded 2 so that it is available when the string as a whole has been processed, Moore introduces a new variable, y, with initial value 0 and updates y according to

$$y \leftarrow f(x, y) = \max(|x|, |y|)$$

Requiring that $y \in [0, 1]$ at the end of processing a string ensures that all moves have performed pushdown stack operations.[2]

Regarding the second question (uniqueness), choosing $x_0 = 1/2$ ensures that each stack state (i.e. each sequence of symbols on the stack of a PDA) places the Dynamical Recognizer at a unique point in $\Re$. But this is not obvious. Consider the case $m = 3$ and $\alpha = 1/10$. If the starting value is $x_0 = 1$, then the stack states e, "3", "33", "333", etc. all generate the same Dynamical Recognizer state, namely $x = 1$. This means the device will fail to distinguish these stack states from each other. I describe an easily assessed condition below (Section 3) that guarantees uniqueness.

To implement complex grammar computation in connectionist networks, it is desirable to use many dimensions. The use of many initially random-valued hidden unit dimensions is crucial to the way gradient descent learning mechanisms like backpropagation solve the symmetry-breaking problem (Rumelhart, Hinton, & Williams, 1986). Moreover, although in any bounded metric encoding of an infinite-state language, the required precision grows exponentially with level of embedding, required precision also grows with the cardinality of the stack alphabet in a one-dimensional recognizer (note Moore's $\alpha \leq 1/(3m + 1)$); use of multiple dimensions can eliminate this growth (see Section 3 below). Also, distributed representations, which have many advantages (analogy, robustness in noise, content-addressability), require multiple dimensions; for learning distributed representations and stack memory with a single learning mechanism, it is desirable to express them both in a similar encoding. The next example illustrates the Dynamical Automaton method of extending Moore's technique to many dimensions.

2.4. Example: Sierpinski Triangle

Figure 2 shows a diagram of the fractal called the *Sierpinski Triangle* (the letter labels in the diagram will be explained presently). The Sierpinski triangle, a kind of Cantor set, is the limit of the process of successively removing the "middle quarter" of a triangle to produce three new triangles.

---

[2] The fact that $push_i$, pop_i and f are all at least piecewise linear in x and y forms the basis of Moore's claim that a dynamical recognizer with all piecewise linear operations can recognize any context free language.

----------Insert Table 2 about here----------

The grammar shown in Table 2 is a context free grammar. This grammar generates strings in the standard manner (Hopcroft and Ullman, 1979; e denotes the empty string). Examples of strings generated by Grammar 1 are "a b c d", "a b c d a b c d", "a b c a a b c d b c d d". The last case illustrates center-embedding. A pushdown automaton for the language of Grammar 1 would need to keep track of each "abcd" string that has been started but not completed. For this purpose it could store a symbol corresponding to the last letter of any partially completed string on a pushdown stack. For example, if it stored the symbol "A" whenever an embedding occurred under "a", "B" for an embedding under "b" and "C" for an embedding under "c", the stack states would be members of {A, B, C}*. We can use the Sierpinski Triangle to keep track of the stack states for Grammar 1. Consider the labeled triangle in Figure 2. Note that all the labels are at the midpoints of hypotenuses of subtriangles (e.g., the label "CB" corresponds to the point, (1/8, 5/8)). The labeling scheme is organized so that each member of {A, B, C}* is the label of some midpoint (only stacks of cardinality ≤ 3 are shown).

----------Insert Figure 2 about here----------

We define a Dynamical Automaton (DA 1), which recognizes the language of Grammar 1, by the *Input Map* shown in Table 3. The essence of the DA is a two-element vector, **z**, corresponding to a position on the Sierpinski triangle. The DA functions as follows: when **z** is in the subset of the plane specified in the "Compartment" column, the possible inputs are those shown in the "Input" column. Given a compartment and a legal input for that compartment, the change in **z** that results from reading the input is shown in the "State Change" column. If we specify that the DA must start with $\mathbf{z} = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$ make state changes according to the rules in Table 3 as symbols are read from an input string, and return to $\mathbf{z} = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$ (the *Final Region*) when the last symbol is read, then the computer functions as a recognizer for the language of Grammar 1. To see this intuitively, note that any subsequence of the form "a b c d" invokes the identity map on **z**. Thus DA 1 is equivalent to the nested finite-state machine version of Grammar 1. For illustration, the trajectory corresponding to the string "a b c a a b c d b c d d" is shown in Figure 3. (**1. a** is the position after the first symbol, an "a", has been processed; **2. b** is the position after the second symbol, a "b" has been processed, etc.)

----------Insert Table 3 about here----------

----------Insert Figure 3 about here----------

As the foregoing example has illustrated, a Dynamical Automaton is like a Dynamical Recognizer except that a partition restricts the function applications. A given symbol can give rise to a given function application only if the automaton is in an appropriate compartment of the partition. This assumption has the consequence that the ungrammaticality of a string is always detected on-line in a Dynamical Automaton but may not be detected until the final symbol has been processed in a Dynamical Recognizer. In this sense, Dynamical Automata are more closely related to on-line connectionist recognizers like the Simple Recurrent Networks (SRNs) of (Elman, 1990, 1991).

In fact, from a representational standpoint as well, the computations of this Dynamical Automaton bear a close resemblance to the empirically observed computations of SRNs. Elman (1991) examined the many-dimensional hidden unit space of an SRN trained on more elaborate recursive languages and found that different lexical classes corresponded to different subregions of the space. Likewise, in the example above, the three lexical classes, A, B, and C correspond to three distinct regions of the representation space (each class has only one member).[3] Elman also noted that the SRN followed similarly-shaped trajectories from region to region whenever it was processing a phrase of a particular type, with slight displacements differentiating successive levels of embedding. Here, the single phrase S is also associated with a characteristic (triangular) trajectory wherever it occurs and slight displacements also differentiate successive levels of embedding.

One can construct a wide variety of computing devices that organize their computations around fractals. At the heart of each fractal computer is a set of iterating functions that have associated stable states and can be analyzed using the tools of dynamical systems theory (Barnsley, 1993[1988]). Hence the name, *Dynamical Automaton*.

3. The general case

The method of Example 2.4 can be extended to all context free languages. I sketch the proof here. The details are provided in the Appendix.

First, the formal definition of a dynamical automaton:

Def. 1. A *dynamical automaton* is a device, M, with the following structure:

$$M = (X, F, P, \Sigma, IM, x_0, FR)$$

(1) The *space*, X, is a complete metric space.

(2) The *function list*, *F*, consists of a finite number of functions, $f_1, \ldots f_K$ where w_i: X $\rightarrow$ X for each i $\in$ {1, ..., K}.

[3] The item "d" does not need a class of its own because its occurrence always puts the computer into a state corresponding to one of the other three classes.

(3) The *partition*, P, is a finite partition of X and consists of compartments, m_1, ... m_K.

(4) The *input list* is a finite set of symbols drawn from an alphabet, Σ.

(5) The *input mapping* is a three-place relation, IM: $P \times \Sigma \times F \rightarrow \{0, 1\}$ that specifies for each compartment, m, and each input symbol, s, what function(s) can be performed if symbol s is presented when the system state is in compartment m. If IM(m, s, f) = 0 for all $f \in F$ then symbol s cannot be presented when the system is in compartment m.

(6) The machine always starts at the *start state*, $x_0 \in X$. If, as successive symbols from an input string are presented, it makes transitions consistent with the Input Mapping, and arrives in the *final region*, FR, then the input string is accepted.

Next, we can define a special class of Dynamical Automata that behave like Pushdown Automata. An easy way to do this is to design dynamical automata that travel around on fractals with several "branches" (the branches of the Sierpinski triangle are its three largest subtriangles). If we define the automata in such a way that (i) the branches can be isolated from one another with linear separators, (ii) each branch corresponds to a stack symbol, and (iii) the top of the stack is always the symbol corresponding to the current branch, then implementation in a connectionist network is facilitated: the network can have a layer of units encoding the position on the fractal; a separate layer of threshold units identifies the top of the stack by determining which branch of the fractal the system is on (see Section 4).

For separation to be possible, the branches must not overlap. The following constructs are useful in this regard.

Def. 2. A *generalized iterated function system* (GIFS) on a metric space X is a set of functions $S = \{X: w_1, ..., w_N\}$ that map X into itself.[4] The system is *uniquely invertible* if each w_i is uniquely invertible on X.

Def. 3. Let $S = \{X: w_1, ..., w_N\}$ be a GIFS on metric space X. Let $\sigma = \sigma_1\sigma_2...\sigma_K$ be a string in $\{1, ..., N\}^*$. Consider the point $x = w_\sigma(x_0) = w_{s_1}(w_{s_2}(...w_{s_K}(x_0)...)) \in X$. The string σ is called an x_0-address of the point x under S. The set of points in X that can be reached from initial state x_0 by finite sequences of applications of functions from *F* is called the *trajectory* of x_0.

Def. 4. Let x_0 be a point in X. If every point in the trajectory of x_0 has a unique x_0-address, then x_0 is called a *cascade point*. In this case, the trajectory, C, of x_0 is called the *cascade*

[4] Barnsley defines an Iterated Function System (IFS) on metric space X as a finite set of contraction maps on X. A GIFS is "generalized" in the sense that the mappings need not be contraction maps--see example 3.2 below.

of x_0. If $x \in C$ and $x \neq x_0$ then the first symbol of the x_0-address of x is denoted $top_C(x)$. If $x = x_0$, we set $top_C(x) = e$, where e denotes the empty string.

Thus each point on a cascade can be mapped to a unique stack state. To make use of this construct, it is necessary to be able to identify cascade points. The following definition and lemma help in this regard:

Def. 5. Let $S = \{X; w_1,..., w_N\}$ be a GIFS on metric space X. A set $O \subset X$ is called a *pooling set* of S if it satisfies the following:

(i) $w_i(O) \cap w_j(O) = \varnothing$ for $i, j \in \{1, ..., N\}$ and $i \neq j$.

(ii) $\bigcup_{i=1}^{N} w_i(O) \subset O$

where $w(O)$ means $\{w(x): x \in O\}$. The set of points in O that are not in $\bigcup_{i=1}^{N} w_i(O)$ is called the *crest* of O.

Lemma 1. Let $S = \{X; w_1,..., w_N\}$ be a uniquely invertible GIFS on metric space X. Suppose $O \subseteq S$ is a pooling set of S and x_0 is in the crest of O. Then x_0 is a cascade point of S.

Proof: See Appendix.

The Lemma makes it easy to identify cascade points of GIFS's.

For example, to establish uniqueness of stack states for the specific example of Moore's context free language recognizer with $m = 3$ and $\alpha = 1/10$ (Example 2.3 above), it suffices to note that that [0, 1] is a pooling set for the GIFS of pop_i's and $push_i$'s and the set $[0, 3/10) \cup (4/10, 6/10) \cup (7/10, 9/10)$ is its crest. Thus any x_0 in this set (Moore used $x_0 = 1/2$) will give rise to a unique mapping for all stack states.

For example, extending and slightly simplifying Example 2.4, if we have a stack alphabet of N symbols, we let $x_0 = \mathbf{1/2} \in \Re^n$ (i.e. the vector in $\Re^n$ with every element equal to 1/2). The function list can then be defined as

$$S = \{ w_n(\mathbf{x}) = 1/2\ \mathbf{x} + 1/2\ \mathbf{e_n}: n \in \{1,... , N\}\}$$

where $\mathbf{e_n}$ is the vector in $\Re^n$ with n'th element equal to 1 and all other elements equal to 0. To see that x_0 is a cascade point for this function list, let O be the open unit hypercube in the positive quadrant of $\Re^n$ with a corner at the origin. For all $\mathbf{x} \in O$, the n'th coordinate of

$w_n(\mathbf{x})$ is in the interval (1/2, 1) and the m'th coordinate of $w_n(\mathbf{x})$ is in (0, 1/2) for $m \neq n$, m, n $\leq$ N. Therefore, each $w_n(O) \subset O$, and $w_n(O) \cap w_m(O) = \varnothing$. Moreover, $\mathbf{x}_0 \notin w_n(O)$ for each n. Thus O is a pooling set for the function system F and, by the Lemma, $\mathbf{x}_0$ is a cascade point.

Thus points on the trajectory of x_0 correspond one-to-one to stack states on the alphabet Σ = {1,... , N}. The current position on the fractal (call it **x**) corresponds the current state of the stack. One can thus define analogs of push and pop moves. In particular, the analog of pushing symbol n is to change the state of the automaton from **x** to $w_n(\mathbf{x})$. The analog of popping symbol n can only legally be performed when the current branch of the fractal is n (this condition can be enforced in the Input Map) and it consists of changing the state from **x** to $w_n^{-1}(\mathbf{x})$.

One can thus define a special class of Dynamical Automata called *Pushdown Dynamical Automata* (PDDAs) which move around on a cascade by making analogs of push and pop moves.

The central claim can thus be stated:

Theorem 1. The set of languages recognized by Pushdown Dynamical Automata (PDDAs) is the same as the set of languages recognized by Pushdown Automata (PDAs).

Proof: See Appendix.

Because the PDDAs described above invoke an extra dimension of representation for every stack symbol, they do not exhibit any growth in required precision as the cardinality of the stack alphabet increases (see discussion of Example 2.3 above). The remainder of this section gives some examples which show how the results above make it easy to design pushdown dynamical automata for various context free languages.

3.1. Example: $l^n r^n$

A simple PDDA recognizes $l^n r^n$ for $n \in \{1, 2, 3,...\}$. Let

$$M = \left(\begin{pmatrix}[0,1]\\[0,1]\end{pmatrix}, \left\{\begin{pmatrix}\frac{1}{2}x_1\\x_2\end{pmatrix}, \begin{pmatrix}2x_1\\x_2+1\end{pmatrix}, \begin{pmatrix}2x_1\\x_2\end{pmatrix}\right\}, P, \{l, r\}, IM, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}\right)$$

Relevant parts of the partition, P, are shown in Table 4a. The input mapping, IM is shown in Table 4b.

----------Insert Table 4 about here----------

In this case, to establish context free language status following the proof of Theorem 1 in the Appendix, we consider the GIFS, S = $\left\{\Re^2; w_1 = \begin{pmatrix} \frac{1}{2}x_1 \\ x_2 \end{pmatrix}\right\}$. Note that $\mathbf{x}_{10} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\mathbf{x}_{20} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ are both cascade points and they have disjoint cascades (condition (i) of the definition of a PDDA). Let these cascades be called C_1 and C_2, respectively. Note that the partition compartment is 1 if the index of **x** is 1 and $\text{top}_{C_1}(\mathbf{x})=e$. It is 2 if the index of **x** is 1 and $\text{top}_{C_1}(\mathbf{x})=1$. It is 3 if the index of **x** is 2 and $\text{top}_{C_2}(\mathbf{x})=1$. It is 4 if the index of **x** is 2 and $\text{top}_{C_2}(\mathbf{x})=e$. Thus the compartment is predictable from the conjunction of the index and the value of top in every case (condition (ii)). The function $\begin{pmatrix} \frac{1}{2}x_1 \\ x_2 \end{pmatrix}$ restricted to C_1 is a push function. The function $\begin{pmatrix} 2x_1 \\ x_2 \end{pmatrix}$ restricted to C_2 is a pop function. The function $\begin{pmatrix} 2x_1 \\ x_2 + 1 \end{pmatrix}$ is equal to $\begin{pmatrix} 2x_1 \\ x_2 \end{pmatrix} \circ \begin{pmatrix} x_1 \\ x_2 + 1 \end{pmatrix}$, i.e., it is the composition of a switch function from C_1 to C_2 and a pop function on C_2 (condition (iii)). Since, moreover, the start and end points are the cascade points of C_1 and C_2 respectively (condition (iv)), M satisfies the definition of a PDDA. Thus, M recognizes a context free language.

In the examples provided so far, the functions of the underlying GIFS's are contraction maps. One may well wonder if cascades only arise in standard iterated function systems (*a la* Barnsley, 1993[1988]) in which all the functions are contraction maps. The following case is an interesting counterexample.

3.2. Example: Crutchfield and Young (1990)
Consider the GIFS, S = {[0, 1]; w_1, w_2} where the functions are given by

$$w_1 = \frac{1}{2}\left(1 + \sqrt{1-x}\right)$$

$$w_2 = \frac{1}{2}\left(1 - \sqrt{1-x}\right)$$

Since w_1 maps the interval O = (0, 1) onto (1/2, 1) and w_2 maps this interval onto (0, 1/2), O is a pooling set of S. Moreover, the point 1/2 is (in) the crest of O. Thus S can be used to record stack states over a two-symbol alphabet by using 1/2 as the start state of a PDDA. However, w_1 and w_2 are not contraction map. For example, $w_2(0.99) - w_2(0.98) > 0.01$.

This example is interesting in part because w_1 and w_2 are the two inverses of the much studied "logistic map", $f(x) = rx(1 - x)$ when $r = 4$. The logistic map has attracted attention because it is a relatively simple (one-dimensional) function with chaotic trajectories (see Stogatz, 1994, for an introduction). Crutchfield and Young (1990, 1994) analyzed f as the generator for a string $\sigma = \sigma_1 \sigma_2 \sigma_3 ...$ where $\sigma_i \in \{0, 1\}$ for all i by considering $f^k(1/2)$ for $k = 1, 2, 3, ...$ and letting $\sigma_k = 1$ if $f^k(1/2) > 1/2$ and 0 otherwise. They found that when $r = 3.57...$ (the so-called "onset of chaos"), the set of initial 2^n-character substrings of σ for $n \in N$ constitutes an indexed context free language. Here, I have taken the opposite tack: inverting a closely-related map introduces an indeterminism that allows us to distinguish histories and thus use the map to model arbitrary grammars (one instance of the inverted logistic suffices for stack alphabets with only two symbols; multiple instances can be used to accomodate more stack symbols). Loosely, one can say that while Crutchfield and Young have analyzed a chaotic map to assess the specific character of its complexity, the present analysis yields a way of using the same map to perform a general class of computations of a similarly complex sort. Given Crutchfield and Young's results, an interesting question is whether there is a canonical grammar associated with each value of r for the inverse logistic map device just referred to. It may be, for example, that for some languages, the corresponding inverse logistic automaton is more tolerant of imprecision in the identification of the final region than for others. I leave this as a question for future research.

4. Implementation in a Connectionist Network

Dynamical Automata can be implemented in connectionist networks by using a combination of signaling units and gating units, some of which may be stochastic. By a *signaling unit*, I mean the standard sort of unit which sends out a signal reflecting its activation state to other units it is connected to. By a *gating unit*, I mean a unit that serves to block or allow transmission of a signal along a connection between two other units. By a *stochastic unit*, I mean one which makes a weighted random choice about which of a finite number of connections to transmit a signal through. All units compute a weighted sum of their inputs and pass this through an *activation function*---either identity or a threshold (a sigmoid can be used in place of both of these with some distortion of the computation due to the nonlinearity).

The use of simple affine functions ($\mathbf{z} \leftarrow q\mathbf{z} + r$) to define the state changes in a Dynamical Automaton makes for a simple translation into a network with signaling and gating units. The coefficients q and r determine weights on connections. The connections corresponding to linear terms (e.g., q) are gated connections. The connections corresponding to constant terms (e.g., r) are standard connections.

----------Insert Figure 4 about here----------

----------Insert Table 5 about here----------

A connectionist implementation of Grammar 1 is shown in Figure 4. Table 5 specifies the weight values and unit types. The network processes strings by representing successive symbols as localist bit vectors on its input layer (the I units) and predicting possible successor symbols on its output layer (the O units) (Elman, 1990). The units z_1 and z_2 form the core of the network. Their values at each point in time specify the coordinates of the current position on the Sierpinski triangle. They have multiple connections going out of and into them. For example, z_1 has three self-connecting loops, one through gate σ_{11}, one through gate σ_{12}, and one through gate σ_{13}. These gates are enabling threshold gates in the sense that if a gate is not activated, then no signal is transmitted along the corresponding connection. The units I_a, I_b, $I_{a,c}$, and I_d are input units. I_b, $I_{a,c}$, and I_d have activation 1 when the input symbol is b, c, or d, respectively. When the input symbol is an a, then one or the other of I_a and $I_{a,c}$ takes on value 1 with positive probability (a stochastic neuron not shown here implements this feature). Each input unit interacts with the z units in two ways: it opens one gate on the self-recurrent connections; and it transmits a weighted signal directly to each z unit. The z units take on activations equal to the weighted sum of their inputs. Although this means that in principle, their activations could be unbounded, all of their computations take place in the bounded region (0, 1) so perfect performance can be approximated by using a quasi-linear (e.g. sigmoidal) activation function that is almost linear in this region. The p units are threshold units that serve to translate the z activations into binary values. The output units, O_b through O_d are threshold units that respond to the p units. Unit O_a happens to need to be on all the time in this grammar so it has no inputs and its threshold is below 0. When an output unit is on, the letter or letters corresponding to it are interpreted as possible next words. A string is deemed grammatical if, at each step of processing, the activated input unit is one of the predicted outputs from the previous step and if the network arrives at the initial state $\mathbf{z} = (1/2, 1/2)$ when the last word is presented.

This network handles ambiguity in the same way that a pushdown automaton does: by guessing. In the example at hand, since "a" is sometimes an ambiguous symbol, and the guesses are evenly distributed, the network has an equal chance of guessing wrong and right each time it encounters an ambiguous "a". Thus, we consider the language generated by the network as the set of strings that it can deem grammatical, even though in a given instance, it may judge any legal string "ungrammatical". Because context often determines the proper interpretation of an ambiguous symbol (e.g. an "a" occuring initially can only be generated by rule 2b), it is possible to use additional neural machinery to constrain the choice to the contextually appropriate one. In this case, a connection from output unit $O_{a,c}$ to the stochastic neuron mentioned above can ensure that a random choice is made between I_a and $I_{a,c}$ is made when $O_{a,c}$ is on and that I_a is activated alone when $O_{a,c}$ is off.

Section 5. Parameter space maps.

One of the main motivations for creating mechanisms for doing complex symbolic computation in metric space computers is that it leads to natural metrics over the symbolic computers themselves. Such metrics may prove useful for understanding how to make complex computers robust under noise, sensitive to statistical properties of an environment,

and learnable from data. In this section, I make some initial explorations into the new perspective afforded on symbolic computers by the metric space implementation described here.

When a dynamical automaton is configured as a pushdown dynamical automaton (PDDA), its functions exhibit precise symmetries in the following sense: it is essential that pop operations have the effect of undoing push operations exactly. One might well wonder what happens if one adopts a more physically realistic perspective and allows the push and pop operations to be only approximate mirrors of one another.

The result can be loss of context-freeness. But the loss is not catastrophic in this case. Instead, the neighboring languages in parameter space take on what one might aptly call "mild context sensitivity".[5]

5.1. Example: Parameterized parenthesis matching

A simple case illustrates. Consider the following parameterized extension of the dynamical automaton for the language $l^n r^n$ which was discussed in Examples 2.2 and 3.1.

$$M = \left(\begin{pmatrix}[0,\infty)\\ [0,1]\end{pmatrix}, \left\{\begin{pmatrix}m_L x_1\\ x_2\end{pmatrix}, \begin{pmatrix}m_R x_1\\ x_2+1\end{pmatrix}, \begin{pmatrix}m_R x_1\\ x_2\end{pmatrix}\right\}, P, \{l,r\}, IM, \begin{pmatrix}1\\ 0\end{pmatrix}, \begin{pmatrix}x_1 \geq 1\\ x_2 = 1\end{pmatrix}\right)$$

where parameters m_L and $m_R > 0$. The relevant part of the partition, *P*, is shown in Table 6a. The input map, *IM*, is shown in 6b.

----------Insert Table 6 about here----------

The scalars, m_L ("Leftward move") and m_R ("Rightward move") are parameters that can be adjusted to change the language the DA recognizes. Figure 5 illustrates the operation of this dynamical automaton. When $0 < m_L = m_R^{-1} < 1$, M recognizes the language $l^n r^n$, $n \in \{1, 2, 3,...\}$.

----------Insert Figure 5 about here----------

We can analyze this automaton as follows. It recognizes strings of the form $l^n r^k$, $n \in \{1, 2, 3,...\}$ where k is the smallest integer satisfying

$$m_L^{\,n}\, m_R^{\,k} \geq 1$$

[5]I thank Jordan Pollack for drawing my attention to this "precarious" quality of context-free dynamical recognizers, and thus motivating the investigation described in this section.

Since we are only considering cases where $m_L > 0$,

$$m_R^{\,k} \geq m_L^{\,-n}$$

$$k \geq \log_{m_R} m_L^{\,-n}$$

$$k = [[\, -n \log_{m_R} m_L \,]]$$

where [[x]] denotes the smallest integer greater than or equal to x.

If m_L is a negative integer power of m_R, then the language of M can be described with a particularly simple context-free grammar. Example 5.1.1 illustrates.

Example 5.1.1. Let $m_L = 1/4$ and $m_R = 2$. Thus $k = [[\, -n \log_2 1/4 \,]] = 2n$. The language is thus $l^n r^{2n}$. This language is generated by a context free grammar with only two rules (Table 7).

----------Insert Table 7 about here----------

If m_L is a non-whole-number rational power of m_R (or vice versa), then the resulting language is still context free, but its grammar is more complicated. Example 5.1.2 illustrates a case like this.

Example 5.1.2. Let, $m_L = 1/4$ and $m_R = 4^{5/6} \approx 3.174802$. Thus $k = [[\, 1.2n \,]]$. The language recognized by this particular parameterization of M is thus $A = l^n r^{[[\, 1.2n \,]]}$. This language is substantially more complicated than the language of the previous example. It requires seven rules in context free grammar format (Table 8). The number of rules grows with the length of the cycle of the coefficient of n.[6]

----------Insert Table 8 about here----------

If m_L is an irrational power of m_R or vice-versa, then M generates a non context-free language. We can show this using the Pumping Lemma for Context Free Languages (Hopcroft and Ullman, 1979, p. 125).

Proposition. The language $A = l^n r^{[[\, qn \,]]}$ for q irrational is not a context free language.

Proof: See Appendix.

---

[6] By the *cycle* of a real number q, I mean the smallest positive integer, p such that pq is an integer.

----------Insert Figure 6 about here----------

These examples show that even one of the simplest parameterized dynamical automata can emulate computing devices with a significant range of complexities. The framework also suggests an interesting new way of examining the relationships between formal languages: we can look at their locations in the parameter space of the dynamical automaton. Figure 6 shows how the simplest (two-rule) context-free languages among $l^n r^{[[-n \log_{m_R} m_L]]}$ are distributed in the first quadrant of $m_R \times m_L$. By adopting a natural metric on this space (e.g. Euclidean distance), we can talk about relationships between languages in terms of distance. On this view, each two-rule language is surrounded by more complex languages (both in the rule-counting sense and the Chomsky hierarchy sense). Although the grammars of the languages near each two-rule language are substantially different from its grammar in possessing large numbers of rules or requiring context-sensitive rules, their distributional properties are rather similar. For example, the language $l^n r^{[[1.01n]]}$ differs from $l^n r^n$ only in very long strings. If we assume that unbiased probabilities are associated with those rows in the Input Mapping of the dynamical automaton that specify transitions out of the same compartment, then the strings on which these two languages differ are quite rare. In Section 6, I speculate how this property may be useful in designing learning algorithms for dynamical automata.

6. Conclusions

I have examined a particular type of computing device, the *Dynamical Automaton*, whose variables and parameters take on real values. In contrast to other work on real-valued automata, which focuses on complexity and tractability issues, I have emphasized the interpretation of real-valued spaces as metric spaces and examined consequences for our understanding of how computing devices are related to one another. First, I identified a class of computing devices called pushdown dynamical automata whose computation is organized around fractals, and showed that this class corresponded to the class of context free grammars. I illustrated a simple method of implementing dynamical automata in connectionist networks. Then I examined a simple dynamical automaton with real-valued parameters and showed how this automaton behaved like various pushdown automata under some parameter settings and like more powerful automata under other settings. Moreover, the different automata were organized in the metric space of the computer's parameters in such a way that nearby automata in the parameter space generated similar sets of strings (in a probabilistic sense). The context free grammars with the fewest rules occupied disconnected regions of the parameter space. Between these were grammars with more rules, including many non-context-free grammars.

6.1 Avenues worth exploring

Several interesting questions are raised by the results described here.

Fleshing-out of Chomsky Hierarchy Relationships. The current proposal to organize languages in a metric space seems, at first glance, to differ substantially from standard complexity-based approaches. However, a close look at the example of Section 5 suggests that the relationship between the current results and the complexity-based results may be one of augmentation rather than revision. In particular, the overarching contrast between context free languages and non-context-free languages is preserved as a contrast between machines with a rational versus an irrational parameter. Two further questions are also worth exploring: (i) Is there a natural way, in the dynamical automaton framework, of defining precisely the recursively enumerable languages, or of defining a set of recursively enumerable languages that contains a set of context free languages as a proper subset? (ii) Do all dynamical automata map standard complexity classes onto independently motivated parameter classes?

Approximations of Infinite Machines. Just as the irrational numbers can be viewed as the limits of infinite series of rationals, so the non context-free devices in the model of Section 5 can be viewed as limits of infinite series of context-free devices. A similar idea has been explored by Crutchfield and Young (1990) and Crutchfield (1994). These authors analyze a particular *indexed context free grammar* as the limit of an infinite series of increasingly complex finite-state devices. They project from their results an approach to signal analysis in which one studies the growth in size of successive machine approximations at one level on the Chomsky Hierarchy in order to find out if a jump to a machine at a higher level is warranted. However, they only explore the case in which bigger and bigger finite state devices approximate a context free device. The current results may thus be useful in extending their method to transitions between higher levels.

Finding all CFLs. The cascade-based analysis makes it possible to identify certain dynamical automata in a parameterized dynamical automaton family which generate context free languages. However, it doesn't necessarily characterize all context free language generators in a particular family. A case in point is the generator described in Section 5: only the simplest two-rule grammars follow a single cascade throughout their computations. It would be desirable to generalize the analysis so that one could identify the entire set of context-free language generators in a given dynamical computing system.

Universality. I have been promoting parameterized dynamical automata as a better way of organizing formal languages than various systems stemming from the Chomsky hierarchy. There is one sense in which the Chomsky hierarchy is more appealing as an organizational tool: it is nearly machine-independent, and thus very universal. The dynamical automaton spaces I define depend on the particular parameterized automaton under study, and do not provide a way of organizing all languages that can be generated using a finite alphabet. Nevertheless, it is interesting to consider the possibility that there might be a privileged set of functions which serves as the basis for a general automaton space to which all dynamical automata can be related. Such a framework might be useful for studying processes that involve incremental search over a very wide range of devices—e.g., evolution, signal identification.

Learning. A natural approach to language learning is to think of it as a process of making small adjustments in a grammar in order to improve predictive accuracy. This approach requires us to define what constitutes a "small adjustment", i.e., to define similarity among grammars. Using standard symbolic formalisms, it is hard to choose among the myriad ways one might go about defining similarity among grammars, especially if the grammars are infinite-state devices. We can examine the number of rules that two grammars have in common. But then the rules that are not shared between the two grammars can have so many forms that it is hard to know how to take them into account. We can assign probabilities to rules and compare the implications for local symbol transition likelihoods. But it is hard to know how to compare between the case in which a rule belongs to a grammar but occurs with very low probability and the case in which the rule is simply absent from the grammar. The hypothesis spaces defined by parameterized Dynamical Automata (as in the example described in Section 5) look promising in this regard. The automata are organized in a continuum in such a way that nearby automata give rise to similar transition behaviors (in a probabilistic sense, as mentioned at the end of Section 5). Thus it may be possible to use dynamical automata as the basis for a gradient descent search without having to make an arbitrary decision about rule cost. Instead of trying to postulate an arbitrary balance between complexity of a grammar and coverage of the data, the learning algorithm can simply search the space for the best-fit automaton.

More generally, metric space computers allow one to see geometric relationships between symbolic computers which are invisible from the standard analytic perspective. It is in this simultaneously microscopic and birds-eye perspective that their strength lies: they reveal the small steps that connect big, ostensibly independent regimes with one another.

Appendix

Lemma 1. Let S = {X; w_1,..., w_N} be a uniquely invertible GIFS on metric space X. Suppose $O \subseteq$ S is a pooling set of S and $\mathbf{x}_0$ is in the crest of *O*. Then $\mathbf{x}_0$ is a cascade point of S.

Proof. (The proof of this lemma follows mainly from the definition of pooling set—Def. 5, in Section 3 of the text.) Suppose, contrary to fact, that there exists $\sigma = \sigma_1\sigma_2...\sigma_J$ and $\rho = \rho_1\rho_2...\rho_K \in \{1, ..., n\}^*$ where $w_\sigma(\mathbf{x}_0) = w_\rho(\mathbf{x}_0)$ but $\sigma \neq \rho$ (recall that $w_\sigma(\mathbf{x}_0) = w_{s_1}(w_{s_2}(...w_{s_K}(x_0)...))$). Let M be equal to the minimum of J and K. Then there are two possibilities to consider. Since the string σ is not equal to the string ρ, either (i) there exists $h \in \{1, ..., M\}$ such that $\sigma_i = \rho_i$ for $i \in \{1, ..., h - 1\}$ and $\sigma_h \neq \rho_h$ or (ii) $\sigma_i = \rho_i$ for $i \in \{1, ... M\}$ and $K \neq J$.

Under case (i), the fact that $w_\sigma(\mathbf{x}_0) = w_\rho(\mathbf{x}_0)$ and the fact that w_1, ..., w_N are one-to-one imply that $w_{s_{-h}}(\mathbf{x}_0) = w_{r_{-h}}(\mathbf{x}_0)$ where $\sigma_{-h} = \sigma_h...\sigma_J$ and $\rho_{-h} = \rho_h...\rho_K$. But $w_{s_{-(h+1)}}(\mathbf{x}_0) \in O$ and $w_{r_{-(h+1)}}(\mathbf{x}_0) \in O$ by condition (ii) of the definition of pooling set. Therefore $w_{s_{-h}}(\mathbf{x}_0) \neq w_{r_{-h}}(\mathbf{x}_0)$ by condition (i) of the definition of pooling set, a contradiction.

Without loss of generality, we can assume that J > K in case (ii). In this case, the fact that $w_\sigma(\mathbf{x}_0) = w_\rho(\mathbf{x}_0)$ and the fact that w_1, ..., w_N are one-to-one imply that $w_{s_{-(K+1)}}(\mathbf{x}_0) = \mathbf{x}_0$. Condition (ii) of the definition of pooling set thus implies that $\mathbf{x}_0$ is in the complement of the crest of *O*. But this contradicts the assumption that $\mathbf{x}_0$ is in the crest of *O*.

Since the assumption led to a contradiction in both cases, it must be the case that $\sigma = \rho$. ∎

The following definitions permit the formal definition of a Pushdown Dynamical Automaton (PDDA) and thus support the proof of the string-generative equivalence of PDDAs and PDAs.

Def. 6 Let S = {X; w_1,..., w_N} be a GIFS with cascade point $\mathbf{x}_0$ and corresponding cascade C. Then w_i:C $\rightarrow$ C is called a *push function* on C.

Def. 7 Let S = {X; w_1,..., w_N} be a a uniquely invertible GIFS with cascade C. The inverses of the w_i are called *pop functions* on C.

The following definition makes it possible to keep track of changes in the control state.

Def. 8 Let S = {X; w_1,..., w_N} be a GIFS. Let C_1 and C_2 be disjoint cascades under S with cascade points $\mathbf{x}_{10}$ and $\mathbf{x}_{20}$ respectively. Then the function f: $C_1 \rightarrow C_2$ such that for all $\mathbf{x} \in C_1$ the $\mathbf{x}_{10}$-address of $\mathbf{x}$ is equal to the $\mathbf{x}_{20}$-address of f($\mathbf{x}$) is called a *switch function*.

Note that f is one-to-one and onto, and hence uniquely invertible for all $\mathbf{x}' \in C_2$.

Def. 9 Let M be a dynamical automaton on metric space X. We say M is a *pushdown dynamical automaton* (PDDA) if there exists a GIFS, $S = \{X; w_1,..., w_N\}$ with cascade points $\mathbf{x}_{10}, \mathbf{x}_{20}, ..., \mathbf{x}_{K0} \in X$, $K \in \{1, 2, 3, ...\}$ and corresponding cascades, $C_1, C_2, ..., C_K$ such that

(i) $C_1, C_2, ..., C_K$ are disjoint.

(ii) For $\mathbf{x} \in \bigcup_{i=1}^{K} C_i$, the partition compartment of **x** is determined by the conjunction of the index, i, of the cascade C_i containing **x** and $\text{top}_{C_i}(\mathbf{x})$.

(iii) Let m be a compartment of the partition of M. Each function $f: m \rightarrow X$ in the input mapping is either a stack function, a switch function, or a composition of the two when restricted to points on one of the cascades.

(iv) The start state, $\mathbf{x}_0$, and final region, FR, of M are contained in $\bigcup_{i=1}^{K} x_{i0}$.

Theorem 1. L(PDA) = L(PDDA).

Proof. In refering to the parts of PDAs, I use the notation of Hopcroft and Ullman (1979).

Part i. (L(PDA) Í L(PDDA)). It is straightforward to prove this direction by converting Moore's (1998) class of Dynamical Recognizers for CFLs into a class of PDDAs for CFLs. Here, for the convenience of someone wanting to implement a PDDA in a connectionist network, I flesh out the method sketched in the text.

Consider context free language L. Derive a PDA, M_a, for L that recognizes strings by final state (Hopcroft and Ullman, 1979).

$$M_a = (Q, \Sigma_a, \Gamma, \delta, q_1, Z_0, F)$$

The goal is to define a corresponding pushdown dynamical automaton,

$$M_d = (X, F, P, \Sigma_d, IM, \mathbf{x}_{10}, FR)$$

Suppose $\Gamma = \{1,..., N\}$, N a positive integer. Let $\mathbf{e}_n \in \Re^N$ be the vector with a 1 on dimension i and 0s on all other dimensions. Suppose $Q = \{1,..., K\}$, K a positive integer. Define the GIFS,

$$S = \{\Re^{N+1}; w_n, n \in \{1, \ldots N\}\}$$

where

$$w_n(\mathbf{x}) = \begin{pmatrix} \frac{1}{2}\mathbf{x}_{[N]} \\ x_{N+1} \end{pmatrix} + \begin{pmatrix} \frac{1}{2}\mathbf{e}_n \\ 0 \end{pmatrix}$$

where $\mathbf{x}_{[N]} = \begin{pmatrix} x_1 \\ \ldots \\ x_N \end{pmatrix}$

Consider the set $C_k = \left\{ x_{k0} : x_{k0} = \begin{pmatrix} \frac{1}{2} \\ \frac{k}{K} \end{pmatrix}, \frac{1}{2} = \frac{1}{2}\sum_{i=1}^{N} \mathbf{e}_i \right\}$ for $k \in \{1,\ldots, K\}$. Then the members of C_k are cascade points of S. to see this, consider $\mathbf{x}_{k0}$ and let *O* be the open unit hypercube in the positive quadrant of $\Re^{N+1}$ with a corner at the origin. Note that for all $\mathbf{x} \in O$, the nth coordinate of $w_n(\mathbf{x})$ is in the interval (1/2, 1) and the mth coordinate of $w_n(\mathbf{x})$ is in (0, 1/2) for $m \neq n$ and $1 \leq m, n \leq N$. Therefore each $w_n(O) \subset O$ and $w_n(O) \cap w_m(O) = \varnothing$. Moreover, $\mathbf{x}_{k0} \notin w_n(O)$ for each n. Thus *O* is a pooling set of S and by Lemma 1, $\mathbf{x}_{k0}$ is a cascade point. Moreover, the cascades $C_1,\ldots, C_K$ corresponding to $\mathbf{x}_{10},\ldots,\mathbf{x}_{K0}$ are disjoint.

Assume, without loss of generality, that $Z_0 = 1$. Let the K ``finite states" of M_a be labeled $q_1, \ldots, q_K$. Assume, without loss of generality, that $q_0 = q_1$.

Now define the parts of M_d as follows. Let $X = X' \times (0, 1]$, where X' is $[0, 1]^N$. Let $\Sigma_d = \Sigma_a$. Let the partition *P* include the compartments $\{X_n \times \frac{i}{K} : n \in \{1,\ldots,N\}$ and $i \in \{1,\ldots,K\}\}$ where $X_n = (0, \frac{1}{2})^N + \frac{1}{2}\mathbf{e}_n$, as well as the compartments $\{\mathbf{x}_{10}\}, \ldots, \{\mathbf{x}_{K0}\}$. Let the name of compartment $X_n \times \frac{i}{K}$ be $M_{i,n}$ for each i and each n, and let the name of compartment $\mathbf{x}_{i0}$ be $M_{i,0}$ for each i.

Build the input mapping, *IM* as follows, where $h, n \in \Gamma\ e \cup \Gamma$, $s \in e \cup \Sigma_a = \Sigma_d$, and $w_e(\mathbf{x}) = \mathbf{x}$:

(i) If (q_i, hn) is a member of $\delta(q_i, s, n)$ then let $(M_{i,n}, s, w_h)$ be a member of *IM*. (push function)

(ii) If (q_i, e) is a member of $\delta(q_i, s, n)$ then let $(M_{i,n}, s, w_n^{-1})$ be a member of *IM*. (pop function)

(iii) If (q_i, n) is a member of $\delta(q_j, s, n)$ then let $(M_{i,n}, s, f(\mathbf{x}) = \begin{pmatrix} \mathbf{x}_{[N]} \\ x_{N+1} + \frac{i-j}{K} \end{pmatrix})$ be a member of *IM*. (switch function)

(iv) Handle the composite cases by composition.

For each, $i \in \{1, ..., K\}$, if $q_i \in F$, then let $\mathbf{x}_{i0}$ be a member of *FR*.

At the beginning of processing, when the state is q_1 and the top of the stack of M_a is e, the current state $\mathbf{x}$ of M_d (namely, $\mathbf{x}_{10}$) has index 1 and $\text{top}_{C_1}(\mathbf{x}) = e$. Thus it can be said at this point that when the state of M_a is q_i and the top of the stack is Z, then the current state $\mathbf{x}$ of M_d has index i and $\text{top}_{C_i}(\mathbf{x}) = Z$. Moreover, the definition of the input mapping implies that this situation never changes during the course of processing a grammatical string. Thus if (q_0, σ, e) leads to (q_j, e, e) under M_a, where $q_j \in F$, then the state $\mathbf{x}$ moves by legal transitions under M_d from $\mathbf{x}_{10}$ to $\mathbf{x}_{j0} \in F$ during the processing of string σ and vice versa. Thus M_a and M_d recognize the same language.

Part ii. (L(PDDA) Í L(PDA)). Consider the PDDA,

$$M_d = (X, F, P, \Sigma_d, IM, \mathbf{x}_{10}, FR)$$

Let the associated GIFS be $S = \{X; w_1, ..., w_N\}$ with cascade points $\mathbf{x}_{10}, \mathbf{x}_{20}, ..., \mathbf{x}_{K0} \in X$ for $K \in \{1, 2, 3, ...\}$ and corresponding cascades, $C_1, C_2, ..., C_K$. Define the corresponding pushdown automaton,

$$M_a = (Q, \Sigma_a, \Gamma, \delta, q, Z_0, F)$$

as follows. Let the stack of M_a be initially empty ($Z_0 = e$) and assume that M_a recognizes strings by emptying its stack. Let $\Sigma_a = \Sigma_d$. For each cascade, C_i, if M_d is on C_i, let the control state of M_a be q_i. Thus define Q as $\{q_i: i \in \{1, ..., K\}\}$. Let Γ be $\{1,..., N\}$. Let δ be defined as follows. Consider $\mathbf{x}$ a point in $\bigcup_{i=1}^{K} C_i$. For each possible index value $i \in \{1, ..., K\}$, and each possible value $Z = \text{top}_{C_i}(\mathbf{x}) \in \{e, 1, ... N\}$, compute the partition compartment to which $\mathbf{x}$ belongs (this is possible under the definition of a PDDA). Suppose this partition compartment is compartment j. Examine the input mapping, IM, for rows containing j. For each such row, (j, s, f) for $s \in \Sigma_d$ and $f \in F$, let δ be defined as follows:

(i) If $f = w_h$ is a push function, then let (q_i, hZ) be a member of $\delta(q_i, s, Z)$.

(ii) If $f = w_h^{-1}$ is a pop function, then let (q_i, e) be a member of $\delta(q_i, s, h)$.

(iii) If f is a switch function that switches from cascade i to cascade l, then let (q_l, Z) be a member of $\delta(q_i, s, Z)$.

(iv) Handle the composite functions by composition.

(v) If $\mathbf{x}_{i0}$ is in the final region of M_d, then let $q_i \in F$.

Note that δ is well-defined in every case because $C_1, ..., C_K$ are disjoint cascades and M_d performs its computations on their union.

Let q_1 be the initial state of M_a (i.e., let the initial state bear the index of the start state of M_d). For $\mathbf{x} = \mathbf{x}_{10}$ it can be truly asserted that if the index of $\mathbf{x}$ is i, the state of M_a is q_i, and if $\text{top}_{C_i}(\mathbf{x})$ is Z, then the top of the stack of M_a is Z. Moreover, this situation is preserved under δ. Thus, if the state, $\mathbf{x}$, moves by legal transitions from $\mathbf{x}_{10}$ to $\mathbf{x}_{j0}$ under M_d during the processing of string σ, then (q_0, σ, e) leads to (q_j, e, e) under M_a and vice versa. Thus M_a and M_d recognize the same language. ∎

Proposition 1. The language $A = l^n r^{[[\,qn\,]]}$ for q irrational is not a context free language.

Proof. I proceed by showing that if $A = l^n r^{[[\,qn\,]]}$ is a context free language, then q is rational.

The Pumping Lemma for Context Free Languages says that if *A* is a context free language, then there is a non-negative integer n such that any string $\sigma \in A$ whose length is greater than n can be written $\sigma = uvwxy$ in such a way that

(i) $|vx| \geq 1$

(ii) $|vwx| \leq n$

(iii) for all $i \geq 0$, uv^iwx^iy is in *A*.

Suppose *A* satisfies the Pumping Lemma for $n > n_0$. Consider a string uvwxy that can be pumped in accord with condition (iii). Clearly, v must consist of a positive number of l's and x must consist of a positive number of r's. Let $c_l = |v|$ and $c_r = |x|$. Without loss of generality, we can assume that v is rightmost in the string of initial l's so $|w| = 0$. Let $d_l = |u|$ and $d_r = |y|$. Then, by the definition of *A* we can write

$$c_r\, i + d_r = [[\, q\,(c_l\, i + d_l)\,]] \qquad (1)$$

For each $i \in N$, let δ_i be the fractional part of $q\,(c_l\, i + d_l)$. Then, by (1), we can write

$$q = \frac{c_r(i+1) + d_r + d_{i+1} - (c_r i + d_r + d_i)}{c_l(i+1) + d_l - (c_l i + d_l)} = \frac{c_r + d_{i+1} - d_i}{c_l}$$

But unless $\delta_{i+1} = \delta_i$ for all i, equation (1) is false for sufficiently large i. Therefore,

$$q = \frac{c_r}{c_l}$$

Since c_r and c_l are integers, q is rational. ∎

References

BARNSLEY, M. (1993[1988]) *Fractals Everywhere.* Boston: Academic Press.

BLAIR, A. and J. B. POLLACK (to appear) Analysis of dynamical recognizers, *Neural Computation.*

BLUM, L., M. SHUB, and S. SMALE (1989) On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, *Bulletin of the American Mathematical Society*, **21**(1):1–46.

BOURNEZ, O. and M. COSNARD. (1996) On the computational power of dynamical systems and hybrid systems. *Theoretical Computer Science*, **168**:417--459.

CRUTCHFIELD, J. P. (1994) The calculi of emergence: Computation, dynamics, and induction. *Physica D*, **75**:11–54. In the special issue on the Proceedings of the Oji International Seminar, Complex Systems–-from Complex Dynamics to Artificial Reality.

CRUTCHFIELD, J. P. and K. YOUNG (1990) Computation at the onset of chaos. In *Complexity, Entropy, and the Physics of Information*, 223–70. Redwood City: Addison-Wesley.

DAS, S., C. L. GILES, and G. Z. Sun (1993) Using prior knowledge in a NNPDA to learn context-free languages. In Hanson, S. J., Cowan, J. D., and Giles, C. L. (eds), *Advances in Neural Information Processing Systems 5*, 65–72. San Mateo: Morgan Kaufmann.

ELMAN, J. L. (1990) Finding structure in time. *Cognitive Science*, **14**:179–211.

ELMAN, J. L. (1991) Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, **7**:195–225.

GILES, C., G. SUN, H. CHEN, Y. LEE, and D. CHEN (1990) Higher order recurrent networks and grammatical inference. In Touretzky, D. (ed) *Advances in Neural Information Processing Systems 2*, 380–7. San Mateo: Morgan Kaufmann Publishers.

HOPCROFT, J.E. and J.D. ULLMAN (1979) *Introduction to Automata Theory, Languages, and Computation.* Menlo Park: Addison-Wesley.

KREMER, S. C. (1996) A theory of grammatical induction in the connectionist paradigm. *PhD Thesis*, Department of Computing Science, Edmonton, Alberta.

MOORE, C. (1998) Dynamical Recognizers: Real-time Language Recognition by Analog Computers. *Theoretical Computer Science* **201**, 99-136.

MOZER, M.C. and S. DAS (1993) A connectionist symbol manipulator that discovers the structure of context-free languages. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, 863–70. San Mateo: Morgan Kaufmann.

PERKO, L. (1991) *Differential Equations and Dynamical Systems*. New York; Springer-Verlag.

POLLACK, J. B. (1991) The induction of dynamical recognizers. *Machine Learning,* **7**:227–252.

RODRIGUEZ, P., J. WILES, and J. ELMAN (1999) A recurrent neural network that learns to count. *Connection Science,* **11**(1), 5-40.

RON, D., Y. SINGER, and N. TISHBY (1996) The power of amnesia. *Machine Learning*, **25**.

RUMELHART, D.E., G.E. HINTON, and R.J. WILLIAMS (1986) Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing, Volume I*, 318-362. Cambridge, Massachusetts: MIT Press.

SIEGELMANN, H. (1996) The simple dynamics of super Turing theories. *Theoretical Computer Science*, **168**: 461–472.

SIEGELMANN, H.T. and E.D. SONTAG (1991) Turing computability with neural nets. *Applied Mathematics Letters*, **4**(6):77–80.

STROGATZ, S. (1994) *Nonlinear Dynamics and Chaos*. Addison-Wesley, Reading, MA.

TABOR, W. (1998) Dynamical automata. *Technical Report* TR98-1694, Cornell Computer Science Department.

TINO, P. (1999/in press) Spatial representation of symbolic sequences through iterative function systems. *IEEE Systems, Man, and Cybernetics, part B: Cybernetics.*

TINO, P. and G. DORFFNER (1998) Constructing finite-context sources from fractal representations of symbolic sequences. *Technical Report* TR-98-18. Austrian Research Institute for Artificial Intelligence, Austria.

WILES, J. and J. ELMAN (1995) Landscapes in recurrent networks. In Moore, J. D. and Lehman, J. F., (eds), Proceedings of the 17th Annual Cognitive Science Conference. Lawrence Erlbaum Associates.

ZHENG, Z., R. M. GOODMAN, and P. SMYTH (1994) Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks,* **5**(2): 320–30.

Acknowledgements
Thanks to Christopher Bader, Stefano Bertolo, Edward Gibson, Robert Jacobs, Dexter Kozen, Jordan Pollack, Paul Rodriguez, Peter Tino, William Turkel, and two anonymous reviewers for inspiring discussions and helpful comments.

Illustration Captions:

Figure 1: A neural network for parenthesis balancing.

Figure 2: An indexing scheme for selected points on the Sierpinski triangle. The points are the analogues of stack states in a pushdown automaton. By convention, each label lists more-recently-added symbols to the left of less-recently-added symbols.

Figure 3: Sample trajectory of DA 1.

Figure 4: Network 1. A neural implementation of Grammar 1. Square nodes denote gating units and circular nodes denote signalling units.

Figure 5: $M(\frac{1}{2}, \frac{17}{8})$ accepting l^3r^3.

Figure 6: The bands in the space m_L x m_R where the simplest (two-rule) context free languages reside.

Table 1: The push and pop functions of Moore (1998).

Name	Function
push_i	$\alpha x + (1-\alpha)\frac{i}{m}$
pop_i	$\frac{1}{\alpha}\left(x - (1-\alpha)\frac{i}{m}\right) = push_i^{-1}(x)$

Table 2: Grammar 1.

S → A B C D	A → a S	B → b S	C → c S	C → a S
S → ϵ	A → a	B → b	C → c	C → a
		D → d S		
		D → d		

Table 3: Dynamical Automaton (DA 1).

Compartment	Input	State Change
$z_1 > 1/2$ and $z_2 < 1/2$	b	$\vec{z} \leftarrow \vec{z} - \begin{pmatrix} 1/2 \\ 0 \end{pmatrix}$
$z_1 < 1/2$ and $z_2 < 1/2$	c, a	$\vec{z} \leftarrow \vec{z} + \begin{pmatrix} 0 \\ 1/2 \end{pmatrix}$
$z_1 < 1/2$ and $z_2 > 1/2$	d	$\vec{z} \leftarrow 2\left(\vec{z} - \begin{pmatrix} 0 \\ 1/2 \end{pmatrix}\right)$
Any	a	$\vec{z} \leftarrow \frac{1}{2}\vec{z} + \begin{pmatrix} 1/2 \\ 0 \end{pmatrix}$

Table 4: a. Part of the partition for the PDDA of example 3.1.

Index	Compartment
1	$\binom{1}{0}$
2	$\binom{(0,1)}{0}$
3	$\binom{(0,1)}{1}$
4	$\binom{1}{1}$

b. The Input Map for the PDDA of example 3.1.

Compartment Index	Symbol	Function
1	l	$\vec{x} \to \binom{\frac{1}{2}x_1}{x_2}$
2	l	$\vec{x} \to \binom{\frac{1}{2}x_1}{x_2}$
2	r	$\vec{x} \to \binom{2x_1}{x_2+1}$
3	r	$\vec{x} \to \binom{2x_1}{x_2}$

Table 5: Weights and unit types for the neural implementation of Grammar 1.

Unit	Type	Input	Weight
σ_{11}	Threshold $= 1/2$	I_a	1
σ_{12}	Threshold $= 1/2$	$I_b, I_{a,c}$	1
σ_{13}	Threshold $= 1/2$	I_d	1
σ_{21}	Threshold $= 1/2$	I_a	1
σ_{22}	Threshold $= 1/2$	$I_b, I_{a,c}$	1
σ_{23}	Threshold $= 1/2$	I_d	1
z_1	Linear	I_a	1/2
		I_b	-1/2
z_2	Linear	$I_{a,c}$	1/2
		I_d	-1
z_1	Linear	z_1 via σ_{11}	1/2
z_1	Linear	z_1 via σ_{12}	1
z_1	Linear	z_1 via σ_{13}	2
z_2	Linear	z_2 via σ_{21}	1/2
z_2	Linear	z_2 via σ_{22}	1
z_2	Linear	z_2 via σ_{23}	2
p_1	Threshold $= 1/2$	z_1	1
p_2	Threshold $= 1/2$	z_2	1
O_a	Threshold $= -1$	—	—
O_b	Threshold $= 1/2$	p_1	1
$O_{a,c}$	Threshold $= -1/2$	p_1	-1
		p_2	-1
O_d	Threshold $= 1/2$	p_2	1

Table 6: a. Part of the partition for the parameterized dynamical automaton of Example 5.1

Index	Compartment
1	$\binom{1}{0}$
2	$\binom{(0,1)}{0}$
3	$\binom{(0,1)}{1}$

b. The input mapping for the parameterized dynamical automaton of Example 5.1.

Compartment Index	Symbol	Function
1	l	$\vec{z} \to \binom{m_L z_1}{z_2}$
2	l	$\vec{z} \to \binom{m_L z_1}{z_2}$
2	r	$\vec{z} \to \binom{m_R z_1}{z_2+1}$
3	r	$\vec{z} \to \binom{m_R z_1}{z_2}$

Table 7: A context free grammar for generating $l^n r^{2n}$.

S	$\rightarrow$	l r r
S	$\rightarrow$	l S r r

Table 8: A context free grammar for generating $l^n r^{[[1.2n]]}$.

$$
\begin{array}{lcl}
\mathrm{S} & \rightarrow & l\ \mathrm{S}_s\ r^2 \\
\mathrm{S} & \rightarrow & l^2\ \mathrm{S}_s\ r^3 \\
\mathrm{S} & \rightarrow & l^3\ \mathrm{S}_s\ r^4 \\
\mathrm{S} & \rightarrow & l^4\ \mathrm{S}_s\ r^5 \\
\mathrm{S} & \rightarrow & l^5\ \mathrm{S}_s\ r^6 \\
 & & \\
\mathrm{S}_s & \rightarrow & \epsilon \\
\mathrm{S}_s & \rightarrow & l^5\ \mathrm{S}_s\ r^6
\end{array}
$$

Figure 1: A neural network for parenthesis balancing.

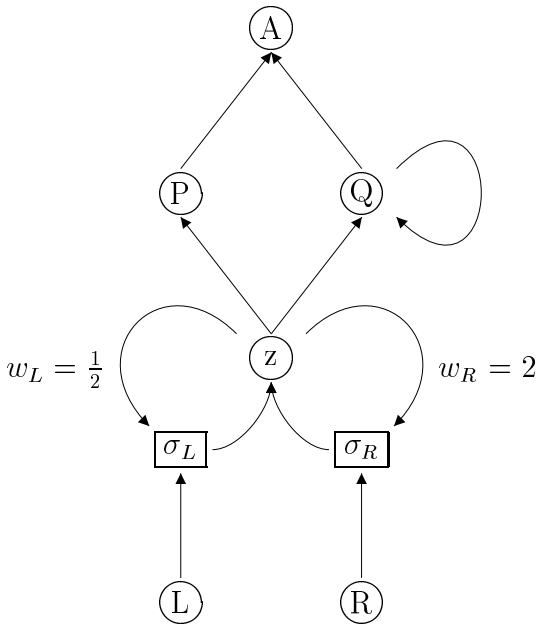

Figure 2: An indexing scheme for selected points on the Sierpinski triangle. The points are the analogues of stack states in a pushdown automaton. By convention, each label lists more-recently-added symbols to the left of less-recently-added symbols.

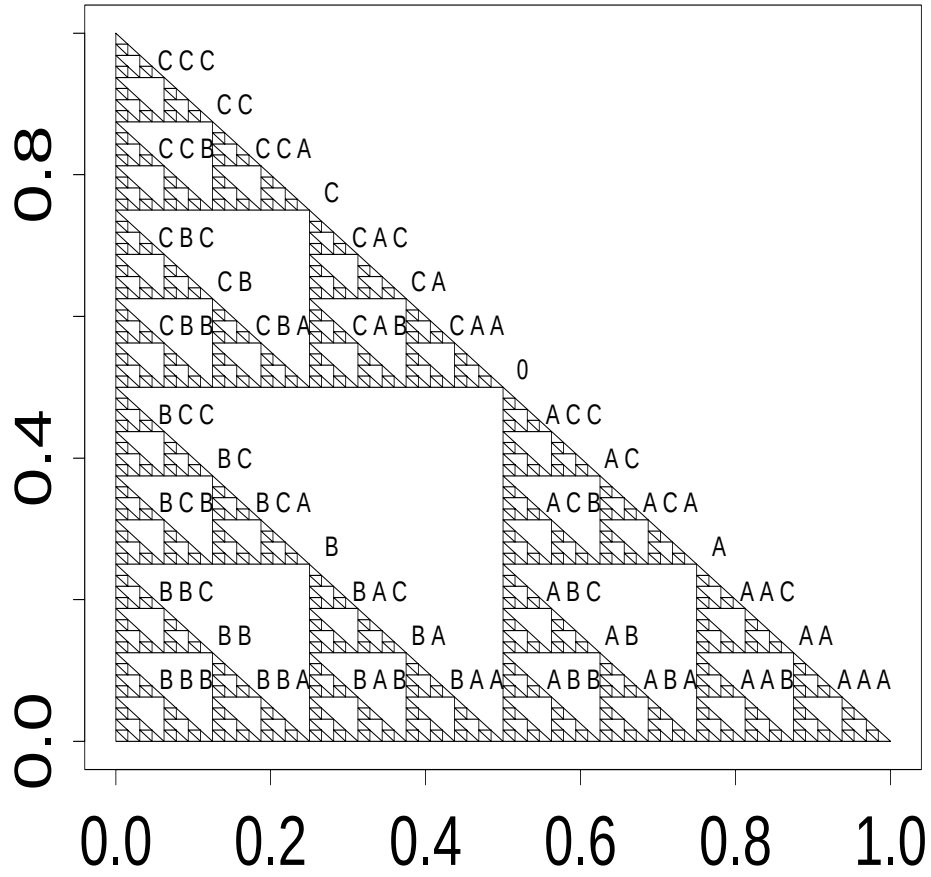

Figure 3: A sample trajectory of DA 1.

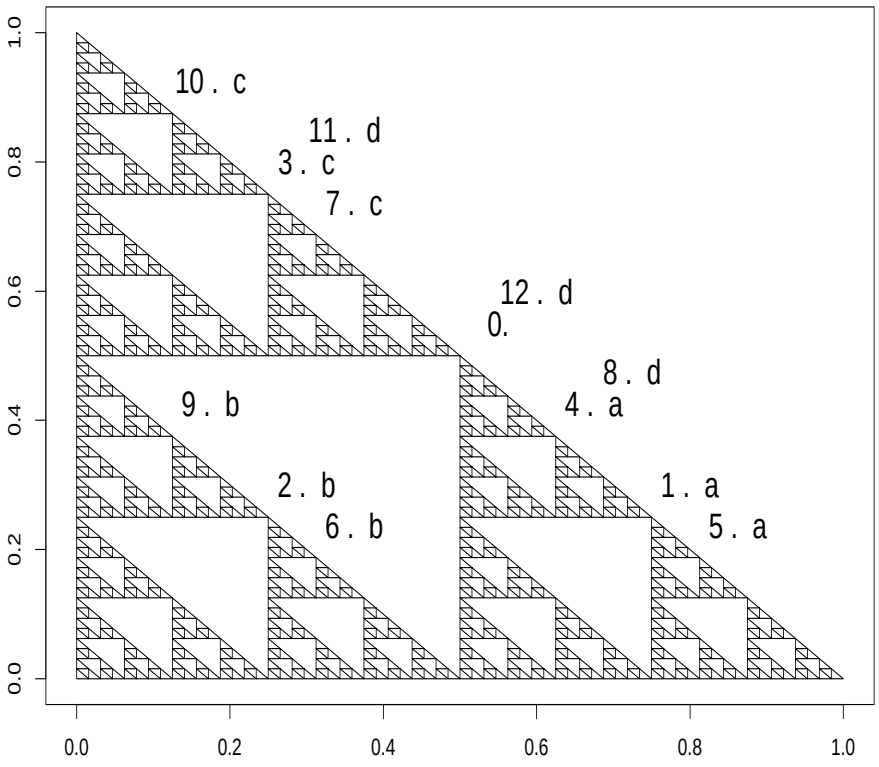

Figure 4: Network 1. A neural implementation of Grammar 1. Square nodes denote gating units and circular nodes denote signalling units.

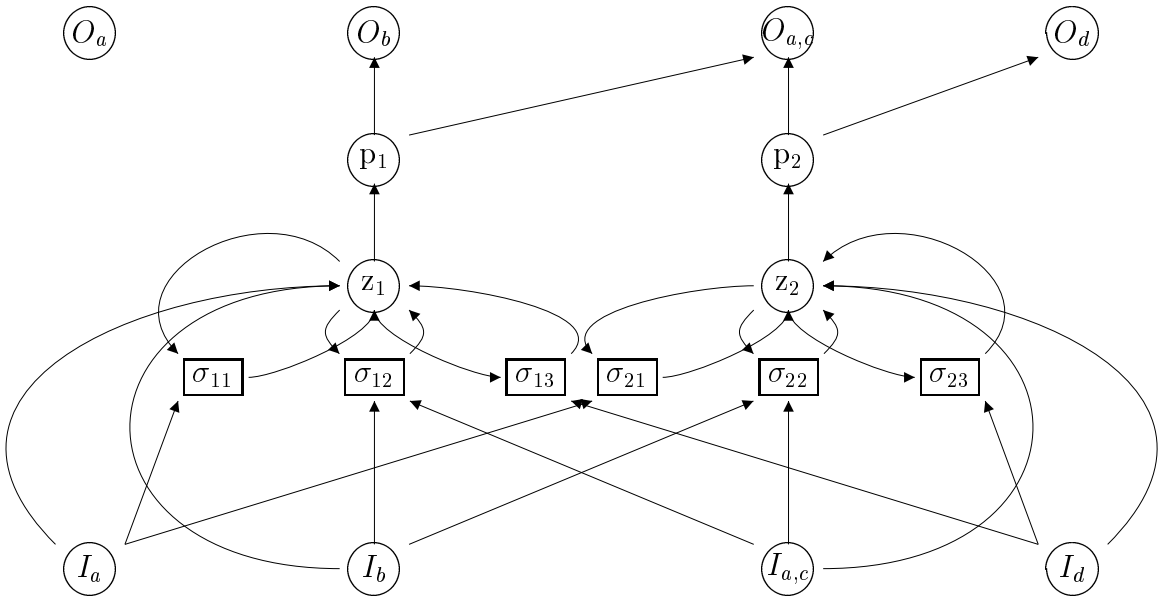

Figure 5: $\mathcal{M}(1/2, 17/8)$ accepting l^3r^3.

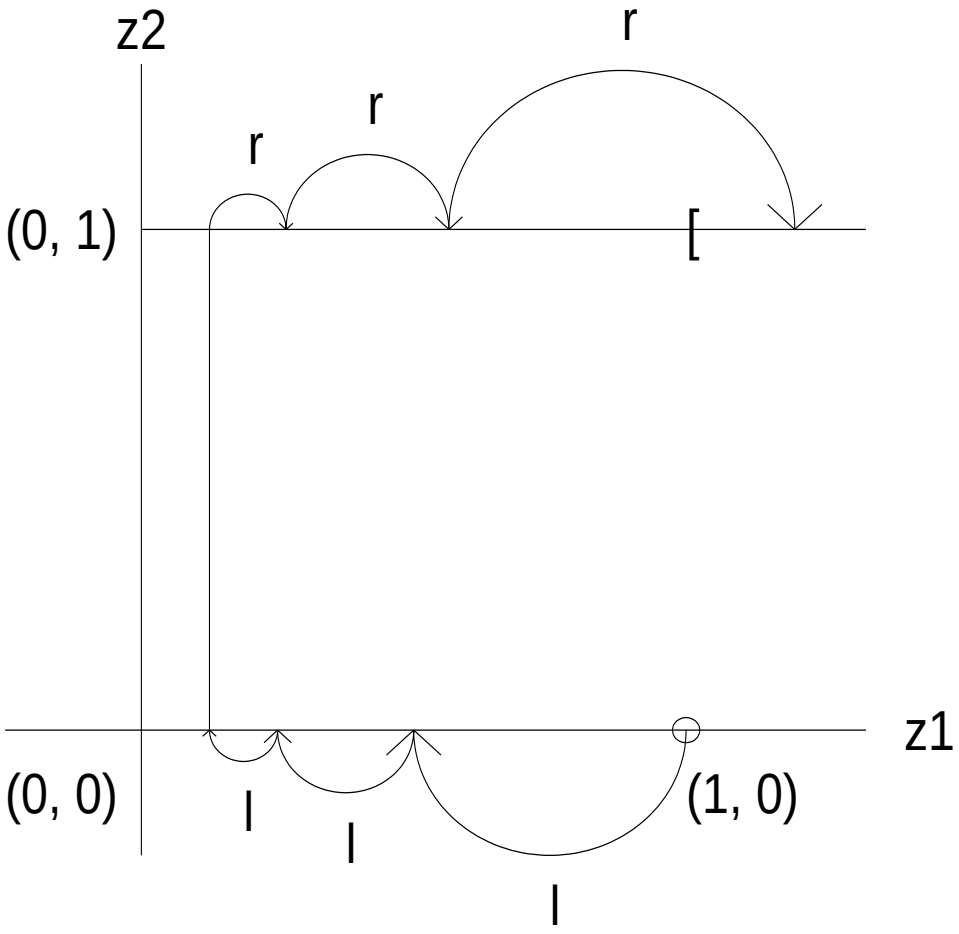

Figure 6: The bands in the space $m_L \times m_R$ where the simplest (two-rule) context free languages reside.