TOPICS
TOPICS IN COGNITIVE SCIENCE

Topics in Cognitive Science 5 (2013) 634–667

ISSN:1756-8757 print / 1756-8765 online
DOI: 10.1111/tops.12036

Fractal Analysis Illuminates the Form of Connectionist Structural Gradualness

Whitney Tabor, Pyeong Whan Cho, Emily Szkudlarek

Department of Psychology and Program in Cognitive Science, University of Connecticut

Received 21 August 2013; received in revised form 6 February 2013; accepted 22 February 2013

Abstract

We examine two connectionist networks—a fractal learning neural network (FLNN) and a Simple Recurrent Network (SRN)—that are trained to process center-embedded symbol sequences. Previous work provides evidence that connectionist networks trained on infinite-state languages tend to form fractal encodings. Most such work focuses on simple counting recursion cases (e.g., $a^n b^n$), which are not comparable to the complex recursive patterns seen in natural language syntax. Here, we consider exponential state growth cases (including mirror recursion), describe a new training scheme that seems to facilitate learning, and note that the connectionist learning of these cases has a *continuous metamorphosis* property that looks very different from what is achievable with symbolic encodings. We identify a property—*ragged progressive generalization*—which helps make this difference clearer. We suggest two conclusions. First, the fractal analysis of these more complex learning cases reveals the possibility of comparing connectionist networks and symbolic models of grammatical structure in a principled way—this helps remove the black box character of connectionist networks and indicates how the theory they support is different from symbolic approaches. Second, the findings indicate the value of future, linked mathematical and empirical work on these models—something that is more possible now than it was 10 years ago.

Keywords: Dynamical systems; Recursion; Symbolic models; Neural (connectionist) networks; Fractal grammars; Simple Recurrent Network (SRN); Generalization; Self-organization

1. Introduction

Various data from language acquisition and historical linguistics suggest that languages change gradually. By a *language*, we mean the verbal behavior of one speaker or a community of speakers and we focus on the formal, patterned nature of such behavior. In

Correspondence should be sent to Whitney Tabor, University of Connecticut, Department of Psychology and Program in Cognitive Science, Haskins Laboratories, 406 Babbidge Drive, Storrs, CT 06269. E-mail: whitney.tabor@uconn.edu

language acquisition, the appearance of syntactic construction types is ordered in a way that suggests incremental progression and new constructions show gradual adjustment of frequencies (e.g., O'Grady, 1997; Wells, 1985); in historical language change, grammatical changes appear to progress by stages through "nearby" construction types, and grammatical changes are again accompanied by relatively smooth frequency changes (Andersen, 1987; Craig, 1991; Ellegård, 1953; Fontana, 1993; Kroch, 1989a; Tabor, 1994).

Traditional grammatical models are symbolic rule systems: They consist of a finite set of rules over a finite alphabet of symbols that generate a (usually countably infinite) set of linguistic structures. There does not seem to be a sensible way of understanding an individual case of such a system as undergoing gradual change of its structure. How, then, are the empirical data in support of gradual acquisition and change to be accommodated?

There are at least three plausible approaches:

[1] At any point in time, each person has a single, symbolic grammar, but at certain, isolated points in each person's life, the person's grammar is replaced by a structurally different one. In a speech community, different people will switch grammars at different times (we assume the switching times are continuously distributed). In this case, averaging over people at a single time gives the impression of intermediate probabilistic behavior, and tracing behavior over time reveals gradual change in these probabilities.

[2a] An individual person stores multiple symbolic grammars and chooses probabilistically among them each time a linguistic decision is made. This approach has been explored in historical language change (Kroch, 1989a, b).

[2b] An individual has a symbolic grammar with probabilistic productions (e.g., a probabilistic context-free grammar). This approach is the standard Variationist conception (Labov, 1972) and is also commonly used in computational linguistics. If the probabilities of the grammar rules change continuously, then the behavior changes continuously.

[3] Connectionist models. These are networks of interconnected units whose input activations stem from environmental stimuli and whose output activations spur actions or specify probabilities of actions. The parameters of the models are (typically) real-valued weights on the connections between units, and the functions that define the interactions between units are (typically) continuous. Thus, continuous change in the parameters produces continuous change in the behaviors. Many studies have modeled language acquisition processes using such connectionist networks (e.g., Elman, 1993; Plunkett & Marchman, 1991; Rohde & Plaut, 1999; Rumelhart & McClelland, 1986). A few have considered historical language change (e.g., Hare & Elman, 1995; Tabor, 1994, 1995).

We think approach [3] has particular promise. We also think it is fundamentally different from approaches [1] and [2]. But it is difficult to assess the empirical validity of approach [3] because many connectionist models are understood via simulation, not via principle (exceptions are the work of Haken—e.g., Haken, 2004—and Smolensky—e.g., Smolensky, 1999), so it is hard to understand what they are claiming in general about data. It is also difficult to tell whether approach [3] does, in fact, make formally or empirically distinct claims from the other two approaches because approaches [1] and [2], on the one hand, and [3], on the other, are typically specified at different levels of description. Under approaches [1] and [2], models are specified via symbolic rules. Under approach [3], models are typically fully described only at a lower, "sub-symbolic" level (Smolensky, 1988). If we described them at the same level of description as approaches [1] and [2], we might discover that they employ the same abstract computational mechanisms as [1] and [2] or we might discover that they do something different.

To this end, we employ tools of dynamical systems theory. A dynamical system is a system that changes. Dynamical systems theory studies functions from state spaces to themselves—typically differential equations or iterated maps, where both the parameter spaces of the functions and the state spaces themselves are connected.[1] Here, we focus on *iterated function systems* (Barnsley, [1988]1993): These are iterated maps with multiple maps on the same state space. We are interested in cases where the maps are continuous. Discrete update, environmentally driven recurrent connectionist networks (e.g., the Simple Recurrent Network ("SRN") of Elman, 1990, 1991) employ such continuous map iterated function systems. The connectedness of the parameter space and the state space are of particular interest. We define the *behavior* of a dynamical system through time in an environment as the map from occurring states of the environment $\times$ state space to successor states of the state space under the dynamics (the environment in this context is a sequence of words). The system can thus travel (or be concentrated) in a subset of the state space and that subset can have a particular form. We are interested in the possibility that arbitrarily small changes in the parameters of the system can give rise to arbitrarily small changes in the state and thus in the form. Viewing the system as a model of language learning or language change, we specify that the parameters change continuously. Because of the connectedness of the parameter space and the continuity of the maps, this assumption puts a strong constraint on change in the form: It must also change continuously (given a particular environment). The state space of the dynamical model is then continuously mapped to a space of probabilities over behaviors. In this way, the continuity of the parameter space can be mirrored in continuity of behavior, which can be observed. Emphasizing the foundation of this continuity, we refer to these models as "connected space computers."

Is this different from symbolic computing? Consider, for comparison, the set of all Turing Machines as a model of a (changing) language. A particular historical state of a language or a particular state of the development of an individual corresponds to a particular Turing Machine with an initial state of its tape. Changing the language or the learner's grammatical system is modeled as replacing the current machine + initial state with a different one. Thus, the parameter space is the space of all possible finite state controllers

× initial tape states. This is a countable set so it is not connected under the natural, discrete topology.[2] The set of processing states possible under the union of all Turing Machines is a countable union of countable infinities and is thus also countable. Therefore, this space too is not naturally connected. Lack of connectedness in the parameter space precludes the possibility of modeling change of behavior as continuous parameter change. Lack of connectedness in the union of state spaces precludes the possibility of continuous change of the state or "form" in the sense referred to above. For these reasons, Turing Machine models (including restricted classes like the set of context-free grammars) do not make the same predictions as connectionist models.

One might think that introducing probabilistic rules in the symbolic domain would change this situation because probabilities, unlike grammars, lie in a connected space. The map from the state of a connected space computer (e.g., the hidden space of the SRN) to a set of probabilities over successor events (e.g., the hidden to output map of the SRN) is analogous. However, introducing probabilistic rules in the symbolic model does not affect the map from the parameter space to the state space under a specified symbol sequence. In particular, it does not change the fact that the rule parameter space and the state space (set of machine configurations visited under the rules) are countable, so it does not make the models capable of making continuity-based predictions of the sort described above. What it does add is a prediction about the statistical distribution of symbol sequences in the environment, and this may be employed to make predictions about the way grammars change in language history and development without employing connected grammar or state spaces.

One might also note that a Turing Machine can approximate any function arbitrarily closely. In fact, when we run connectionist models on digital computers, we are making use of such approximations. Does this not imply that a Turing Machine model can come arbitrarily close to a connectionist model and is thus not empirically distinguishable from it in any way that we should care about empirically? It is possible, but there are some challenges that need to be addressed for this approach to work. We return to this point in the Conclusions.

We focus on a case that highlights the distinction between the two approaches: recursion learning. Under traditional, symbolic formulations, recursion involves precise coordination among multiple symbolic rules. Since the symbolic rules themselves cannot morph continuously, and all the rules have to be in place for the recursion to work, it is hard to envision how a recursive system could gradually take form. A helpful idea provided by dynamical systems theory is that recursive computation can take place in connected space computers using fractals (Moore, 1998; Rodriguez, Wiles, & Elman, 1999; Rodriguez, 2001; Siegelmann, 1999; Tabor, 2000; Wiles & Elman, 1995). Fractals are sets that are self-similar (or nearly self-similar) at arbitrarily small scales (Barnsley, [1988]1993). A fractal is a kind of recursive spatial object. A key insight offered by these studies is that one can use the spatial recursion of fractals to keep track of the temporal recursion of complex symbol sequences. It is easier to understand how fractals can come into being gradually than it is to understand how rule systems can come into being gradually. Thus,

the fractal models offer a way of predicting some of the gradual change phenomena mentioned above.

Pollack (1987, 1991) anticipated the major insights: connectionist networks can learn to process recursive structure, they do it using iterated function systems, traveling on fractals (a la Barnsley, [1988]1993), they go through phase transitions on the way to achieving complex memory structure (e.g., stack memory). Servan-Schreiber, Cleeremans, and McClelland (1991) anticipate our discussion of *continuous structural metamorphosis* with their notion of a *graded state machine*—we make the concept more precise by studying the formation of the fractals.

A number of computational results from the 1990s and early 2000s provide helpful background. Connectionist networks not only do Turing computation but they also do super-Turing computation (e.g., Moore, 1998; Siegelmann, 1999), the models can fairly easily learn finite samples from $a^n b^n$ (a counting recursion/linear state growth language)[3] and generalize to higher levels (Blair & Pollack, 1997; Wiles & Elman, 1995), some can do counting recursion with context-sensitive languages (Bodén & Wiles, 2002; Bodén & Blair, 2003; Chalup & Blair, 2003; Rodriguez, 2001); they do this by traveling on fractal sets, as predicted (Rodriguez, 2001)—see Kolen and Kremer (2001) for review.

This work is helpful, but it has largely subsided and its relevance to central problems in cognitive science has not been recognized. We see three probable reasons for this: (1) counting recursion is not very natural language-like (Corballis, 2007), and training was not very successful with mirror recursion and other exponential state growth languages (Rodriguez, 2001); (2) the learning work is not very mathematical, so it is hard to see how what it is claiming precisely and in general; (3) all of this work is not human empirical so there is a lack of evidence that it bears on problems in human cognition.

Our previous work (Tabor, 2003, 2011) helps with (1) by introducing the fractal learning neural network (FLNN), a connectionist model that can implement exponential state growth languages exactly and can learn to approximate them with high accuracy. The most significant feature of this work is that it gives us a glimpse of a connectionist network learning a symbol sequencing problem with complexity much closer to that of natural language than the previous cases. We present the network in comparison to alternative ways (1 and 2 above) of modeling gradual structure change within the symbolic paradigm and find evidence that the network is doing something different. We hope our results will spur future research of two kinds: one, mathematical investigation of the nature of the computational change that occurs during the continuous metamorphosis; the other, empirical investigation of the correspondence between the theory and human behavior. In the past 25 years, there have been significant developments in our understanding of computation on the reals and other connected spaces (e.g., Blum, Shub, & Smale, 1989; Blum, Cucker, Shub, & Smale, 1998; Calvert, 2011; Moore, 1996, 1998) and in our understanding of the computational properties of real numbers (e.g., Downey & Hirschfeldt, 2010; Nies, 2009) that may make the mathematical development more feasible. Likewise, there have been significant developments in our ability to study the fine-grained properties of human language perception and learning (e.g., Altmann & Kamide, 1999; Cho, Szudlarek, Kukona, & Tabor, 2011; Magnuson, Tanenhaus, Aslin, & Dahan, 2003; Misyak,

Christiansen, & Tomblin, 2010; Tanenhaus, Spivey-Knowlton, Eberhard, & Sedivy, 1995) that make that make the experimental development more feasible.

Our purpose in this article is to review two connectionist models of recursion learning and to demonstrate the gradual formation of fractal structure in both of them. The first type is the FLNN. These systems afford mathematical analysis of a relatively simple sort and thus help make the ideas perspicuous. On the other hand, it is not clear how general FLNNs are—they have not yet been shown to work in all the kinds of cases that seem relevant for natural language. The second type we investigate is the SRN (Elman, 1990). This model has been successfully trained on a much wider variety of distributions than the FLNN and has been widely studied as a model of psychological phenomena. We present evidence that an SRN trained on a recursive language uses fractal organizational principles, somewhat like, though not exactly like, the FLNN—this analysis extends previous results on fractal organization in connectionist networks (e.g., Pollack, 1991; Rodriguez, 2001; Tabor, 2000, 2003) by offering precise methods of measuring the degree of fractal organization in high dimensional state spaces.

1.1. Formal devices

Two devices from classical computing are particularly relevant: a *finite state machine* (FSM) is a computer that can only be in a finite number of distinct states; a *pushdown automaton* (PDA) is a finite state machine combined with a pushdown stack—that is, an unbounded, first-in, last-out symbol string memory. The last symbol added to the stack is called the *top of stack*. The states of the finite state part of the PDA are sometimes called *control states*. The rules of processing of a PDA specify how the stack and the control states change given the current input, the current top of stack, and the current control state. A *formal language* is a set of finite-length strings drawn from a finite alphabet. We say that a machine *recognizes* a formal language, *L*, if, for any finite length string, it can decide, in a finite number of steps, whether the string belongs to *L*. From the definitions just given, it is clear that every FSM language is a PDA language. But there are infinite-state PDA languages—they require the PDA to employ stack states of arbitrary length—and these cannot be recognized by any FSM (Hopcroft & Ullman, 1979).

Several properties of standard connectionist networks are also relevant: typically, connectionist networks are organized into successive layers of units. The first layer detects the input and sends the signal to a "higher" layer, which may interact with itself for a time (recurrent connections), and then send the signal to an even higher layer, etc. The units in these layers are typically simple neural detectors with information processing characteristics related to types of neurons that have been observed in the brain: One common type is a *sigmoidal* or *threshold* unit—these units have the property of linearly separating the activations of the layer below into two halves: if the activation pattern on the lower layer is on one side of a hyperplane that divides the space, the sigmoid/threshold unit turns off; if the pattern on the lower layer is on the other side of the hyperplane, the sigmoid/threshold unit turns on (Minsky & Papert, 1988[1969]). Another common type, also evidenced in the brain, is an *on-center-off-surround* or *radial basis function* unit.

These units turn on when the pattern on the lower layer is in a particular spherical (or elliptical) region of the lower space, and they turn off otherwise.

1.2. Overview

The remainder of the article is organized as follows. Section 2 considers an example of a PDA language that is not an FSM language, defines "fractal grammars," and shows how a fractal grammar keeps track of the infinite structure in the example language. Section 3 describes the FLNN—a network that can learn the language of a fractal grammar—and shows the result of training it on the language of Section 2. Section 4 considers a slightly different PDA language that has not been successfully learned by an FLNN but is more suitable for training people. An SRN is trained on sample sentences from this language and assessed to see whether it shows similar representational and learning characteristics to the FLNN. Section 5 concludes.

2. Fractal learning neural networks

2.1. Dynamical automata

To formalize connected space computing, Tabor (2000) defines dynamical automata (see the similar formalism of Moore, 1998). The essential idea is that there is a complete metric space with a set of functions that map parts of the space to other parts of the space.[4] Each function is associated with some symbols from a finite alphabet that license it. Thus, if the system starts in the domain of one of the functions, and receives one of the symbols that license the function, then it can apply the function, bringing it to a new part of the space, where a different range of function applications and possible symbols may be available. We can define a particular subset of the space as the *Final Region*. When the system reaches the Final Region, we consider the sequence of symbols it has processed since it was at the initial state, a sentence. The set of sentences that the automaton recognizes from a given initial state is the *language* of that automaton-state pair. Similarly, we can start the system in some initial state and let it generate sequences of symbols by selecting among the legal options at any point. As in classical computing, the set of generated sentences is the set of recognized sentences, and we will henceforth refer to this set as the sentences "processed" by the dynamical automaton. Thus, a dynamical automaton, in combination with a particular initial state, specifies a language of strings as in classical computational theory (Hopcroft & Ullman, 1979). In fact, the framework is identical to that of the classical theory with the sole difference that the space is a connected (uncountable) space rather than a countable set. This difference changes the kinds of languages that can occur (Moore, 1998; Siegelmann, 1999; Tabor, 2009).[5] It also draws attention to the metric relations among the states, and these turn out to be revealing.

More formally, Tabor (2000) defines a *Dynamical Automaton*, *DA*, by

$$DA = (H, F, P, \Sigma, IM, x_0, FR) \tag{1}$$

H is a complete metric space (Barnsley, [1988]1993; Bryant, 1985). F is a finite list of functions $f_i : H \rightarrow H$, P is a partition of the metric space, Σ is a finite symbol alphabet, *IM* is an *Input Map*—that is, a function from symbols in Σ and compartments in P to functions in F. The input to the dynamical automaton is a string of symbols. The machine starts at x_0 and invokes functions corresponding to the symbols in the input in the order in which they occur, if it is possible to do so under the constraints of the input map. If every symbol presentation conforms to the Input Map and, when the last symbol has been presented, the system is in the region $FR \subseteq H$, then the DA *accepts* the string. As mentioned above, dynamical automata can process many types of languages. Tabor (2000) shows how to implement any context-free language in a dynamical automaton.

2.2. A sample fractal grammar

Some dynamical automata navigate on fractal sets. In this case, they may be called *fractal grammars*. Tabor (2003) considers the language L1, identified in Table 1. L1 is a probabilistic context-free language the sentences of which cannot be processed by a finite state machine.[6] This language has a particularly simple PDA implementation: Even with every input symbol evoking a single stack event (push or pop in this case), no control state changes are necessary; the PDA can do all of its symbol processing by referring to the top of stack. Table 2 specifies PDA1, a PDA for L1.

Table 3 shows a dynamical automaton encoding of L1. To see how this encoding works, it is useful to compare DA1 to PDA1. Under PDA1, there is a distinct stack state for every grammatical sequence of a's, x's, b's, and y's that has a distinct future—that is, for every *causal state* in the sense of Crutcheld (1994).[7] In the notation of Table 2, the set of all stack states is {A, X, AA, AX, XA, XX, AAA, AAX, AXA,...}. Fig. 1A shows a way of organizing these states in a metric space by mapping them to a fractal. DA1 is designed to make use of these stack states to keep track of recursive dependencies in Language L1. The dynamical automaton always starts at $\binom{0}{0}$. It stores the history of its current

Table 1
PCFG1, a probabilistic context-free grammar that specifies Language L1. This language is closely related to the language S → a (S) b, S → x (S) y, a palindrome language. It differs in allowing recursive embedding after any symbol, not just a phrase-initial symbol. Optional constituents (indicated by parentheses) occur with probability 0.2

0.5	S	→	A B
0.5	S	→	X Y
1.0	A	→	a (S)
1.0	B	→	b (S)
1.0	X	→	x (S)
1.0	Y	→	y (S)

Table 2
PDA1, a pushdown automaton for L1. For each row of the table, the "Input Symbol" value shows the allowable language symbol that can occur next when the top of the stack has the value in "Top of Stack." The "Stack Action" value specifies what change occurs to the stack when the symbol is observed/generated. "Push i" means "add the symbol i to the top of the stack." "Pop" means remove the top of stack. "$R \rightarrow i$" means "replace the symbol R with the symbol i." The "Probability" value shows the probability of the specified action given the top of stack. When the Stack Action is "End," the current symbol sequence is deemed a complete sentence. Initialized with the stack consisting of just the symbol R, PDA1 generates the same (dendogram) language as PCFG1

Probability	Top of Stack	Input Symbol	Stack Action
0.5	R	a	R→A
0.5	R	x	R→X
0.1	∅, A, X	a	Push A
0.1	∅, A, X	x	Push X
0.8	A	b	Pop
0.8	X	y	Pop
0.8	∅	None	End

Table 3
The Input Map for DA1. The initial state is $\binom{0}{0}$. For each row of the table, if the system lands in the region indicated under Compartment, then the symbol(s) indicated under Symbol may be processed via application of the state change specified under function

Compartment	Input	State change
Any	a	$\vec{z} \leftarrow \frac{1}{2}\vec{z} + \binom{-1}{-1}$
Any	x	$\vec{z} \leftarrow \frac{1}{2}\vec{z} + \binom{-1}{1}$
$z_1 < 0$ and $z_2 < 0$	b	$\vec{z} \leftarrow 2(\vec{z} + \binom{1}{1})$
$z_1 < 0$ and $z_2 > 0$	y	$\vec{z} \leftarrow 2(\vec{z} + \binom{1}{-1})$

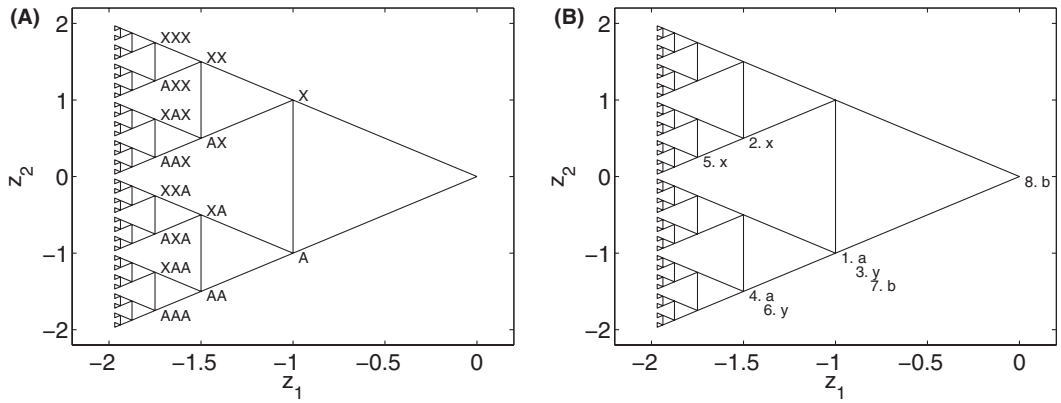

Fig. 1. (A) The correspondence between DA1 states and stack states. Stack states (labeled) are associated with the apices of rightward-pointing triangles. Each letter in a label identifies a symbol on the stack. The top of the stack is the right-most letter in each label. (B) The trajectory associated with the sentence, "a x y a x y b b," from Language 1. "1" identifies the first word, "2" the second, etc.

state in a vector. When the pushdown automaton would perform a push operation (i.e., on observing a or x in the example language), the dynamical automaton shrinks its current state by a factor of 2 and adds the result to a "base vector" corresponding to the new symbol. The base vector for a is $\binom{0}{-1}$. The base vector for x is $\binom{0}{1}$. We can also think of the origin ($\binom{0}{0}$) as a base vector corresponding to empty stack. When the pushdown automaton would perform a pop (observing b or y) in the current system, the dynamical automaton subtracts the current base vector and doubles the result. This restores the system to the state it was in before the last push. Fig. 1B shows the sequence of states visited by the dynamical automaton when it processes a sample center-embedded sentence. Under this scheme, the nearest base vector to the current state is a consistent indicator of top of stack (A, X, or empty). The point of the scheme is as follows: in the next section, we will describe a connectionist network whose first-hidden-layer state travels on a fractal of this sort. The next higher layer of this network will have the job of determining which stack state the system is currently in. Because we have organized the hidden space so that all the processing states that have the same top of stack are located in the same corner, or on the same side, of the space, it is a simple matter for the next layer of the network to detect the current top of stack using sigmoidal (linear separator) or radial basis function (spherical separator) units. In this language, if the network can accurately determine the top of stack, then it can accurately predict the future of any grammatical sequence.

It is useful to consider the set visited by the dynamical automaton if it is started in its initial state and driven by the output of the dendogram language L1. This set is precisely the fractal. In general, following Tabor, Juliano, and Tanenhaus (1997), we call a set generated in this fashion the *visitation set* of the device: It may or may not be a fractal (Tabor, 2009). In practice, we will often examine finite samples of the model's visited set to probe the structure of its system. Where it does not cause confusion, we will also refer to these finite samples as *visitation sets*.

The proofs of formal equivalence between context-free grammars and relevant subclasses of dynamical automata in Moore (1998), Siegelmann and Sontag (1991), and Tabor (2000) are constructive proofs, and they all use fractal grammars. Thus, this formalism is quite usable for implementing grammars of the sort commonly used in formal studies of online parsing, language learning, and language change.

2.3. Architecture and activation dynamics of the fractal learning neural network

The previous section defined some formal mechanisms that support the processing of context-free languages with neural devices. Tabor (2000) shows how one can hand wire a full, layered implementation. Here, we are particularly interested in learning, so we turn to the FLNN (Tabor, 2003), a type of layered network that can learn some relatively complex recursive languages and whose resulting encodings are particularly amenable to analysis. We illustrate the behavior of an FLNN by training it on data from language L1.

One form of the FLNN architecture is shown in Fig. 2 (from Tabor, 2011). The input layer receives successive indexical bit vectors (one unit activated at 1, the rest 0) specifying symbols, as indicated by the labels below the input units. The first hidden layer has a

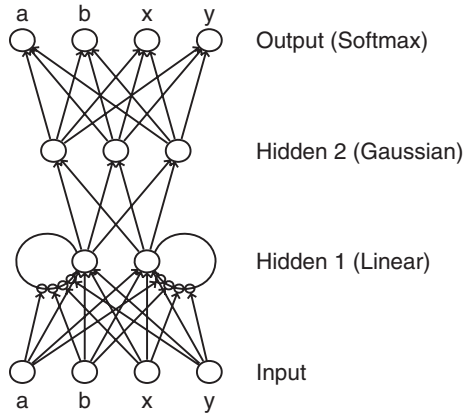

Fig. 2. An FLNN architecture suitable for learning language L1.

linear (identity) activation function. All maps are discrete. The following is a general form of the update rule for linear hidden units:

$$z_i(t) = \sum_{j \in Inputs} w_{ij} a_j(t) + \sum_{k \in Hidden1} \sum_{j \in Inputs} s_{ikj} z_k(t-1) a_j(t) \tag{2}$$

Here, t indexes time, a_j is the activation of the j'th input unit, and z_i is the activation of the i'th first hidden layer unit; w_{ij} is the (first-order) weight on the connection from unit j to unit i; s_{ikj} is the (second order) weight from input unit j and the previous state of hidden unit k to the current state of hidden unit i. Tabor (2000) shows that only the self-weights need to be manipulated to implement any context-free grammar and it suffices to use the same contraction/expansion factor across all hidden dimensions for a given word, one can set all the non-self-connections in Hidden1 to 0 and define $s_j = s_{iij}$. Moreover, because the input vectors are indexical bit vectors, the description can be further simplified by defining $z_i(t, j)$ to be the value of $z_i(t)$ when unit j is the activated input:

$$z_i(t,j) = w_{ij} + z_i(t-1) s_j \tag{3}$$

In Eq. 3, s_j implements the fractal scaling (pushing and popping), and w_{ij} specifies the i'th element of the "base vector" for input j (compare Table 3).

The Linear units have first-order connections to the second hidden layer, which has (Gaussian) radial basis function units:

$$g_i(t) = exp\left[-\frac{|\vec{w}_i - \vec{z}(t)|^2}{b_i^2} \right] \tag{4}$$

Here, g_i is the activation of the i'th second hidden layer unit and b_i^2 is its "bias" (a parameter that controls the radius of the region of Hidden1 space over which the unit is

active); $\vec{w}_i$ is the vector of weights feeding from the Linear Units to unit i of the Guassian units; $\vec{z}$ is the vector of Linear Unit activations; $|\ldots|$ is the Euclidean norm.

As noted above, a system for identifying stack states on the basis of a fractal like that shown in Fig. 1A must be able to distinguish the two branches of the fractal from one another and from the apex of the fractal. In the FLNN, the Gaussian units serve this purpose—each will evolve during the learning process so that it is centered on the appropriate part of the fractal. In principle, the same end could be accomplished using only the output sigmoidal units, and indeed, the SRN described in the next section appears to accomplish something like this, but in experimenting with the FLNN, we were able to achieve successful learning only when a layer of radial basis function units was included in the architecture. Giving the network a unit for each locus (even though the loci are initially random placed—see below) may help it stabilize the tri-partite structure (see Tabor, 2011).

Finally, the Gaussian units have first-order connections to the output units, which, as a group, have the normalized exponential ("soft-max") activation function, since they model the probability distribution for the next symbol at each point in time ($o_i(t)$ is the activation of the i'th output at time t):

$$o_i(t) = \frac{exp(net_i(t))}{\sum_{k \in Outputs} exp(net_k(t))} \tag{5}$$

$$net_i(t) = \sum_{j \in Hidden2} w_{ij} g_j(t) \tag{6}$$

2.4. Training procedure

The network was started with some of its weights in an unbiased state and some in a randomly biased state. The weights from the input to the linear hidden layer were set to 0 (unbiased). Since the linear hidden self-weights are for implementing contraction and expansion in the fractal, their unbiased value is 1 and they were set to that value initially. The Linear and Guassian units need to undergo symmetry breaking since within each of these layers, the units have to play different roles, so the Linear → Gaussian and the Gaussian → Output weights were initialized to random values uniformly distributed on [−0.3, 0.3]. The Gaussian biases (which specify the radii of the on-center-off-surround spheres) were clamped (arbitrarily) at 0.25. The Softmax biases were clamped at 0.

The training was accomplished in corpus-batch mode, using a hill-climbing procedure. We used a corpus consisting of all distinct grammatical sentences up to length 6 from L1. These were sampled uniformly. The corpus thus contained sentences with 0, 1, and 2 levels of embedding only. (We will refer to these henceforth as "Level 1," "Level 2," and "Level 3" sentences, respectively). We did not use a random sample from the dendogram language because we wanted to reserve sentences with more than two levels of

embedding for testing. With the weights at their initial values, the entire corpus was fed in sequence to the network and, at each word, the layers of the network were updated in the sequence, Input, Linear, Guassian, and Output. The target for each word was the next symbol distribution, given the current causal state. Error on a particular word was measured as Kullback–Leibler Divergence at the output layer ($E = \sum_i t_i log \frac{t_i}{o_i}$, where t_i is the probability of word i in the current context and o_i is the activation of output unit i). The error was summed over all the words in the corpus to determine the *current error value*. Then, the process was repeated for each member of a set of points on a sphere in weight space around the current point (radius 0.01). Only basis directions and their negatives were considered. If one of the weight changes produced lower error, then that new value became the current error value and the process was repeated. This gradient method was iterated until the mean error per word dropped below 0.001 or no investigated direction produced a lower error than the current error value.

Matlab code for running this process with L1 and another language (Tabor, 2011) is available at http://psychology.uconn.edu/labs/solab/papers.html.

2.5. Training results

To test the prediction accuracy of a trained network, we considered all the sentences in the training corpus. For each such sentence, we set the linear hidden units to $\binom{0}{0}$ and presented the sentence, asking at each juncture between words whether the highest activated unit on the network's output layer was among the grammatical possibilities given the training grammar and the sequence of symbols previously observed. We counted the word as correctly processed if the answer was "Yes." We counted a sentence as correctly parsed if every word in it was correctly processed. This is a slightly less stringent test than the now standard method of comparing the network's output activations, interpreted as a probability distribution, to the distribution of next symbol possibilities as determined by the training language, but it has the advantage of providing a definitive identification of grammatical failure (because we count the output as right or wrong, rather than providing a continuous measure). We chose this method because we were interested in whether the network had mastered the structure of the language.

The FLNN training algorithm often finds a globally optimal solution, so we only ran one network and performed further testing on this network. The result of testing the training corpus in the manner described above was that every sentence was correctly parsed. We then considered a test corpus consisting of all sentences with four concentric levels (three levels of embedding) and another consisting of all sentences with five concentric levels. In the four levels test, the network parsed 79 out of 80 sentences correctly. In the five levels test, the network parsed 414 of 448 sentences correctly. These results indicate that the network generalized fairly successfully but did not learn exactly the language L1. It is important to note that the errors that the network makes are not the result of noisy processing—there is no noise in the network's choices in this test. The errors stem from the structure of the network's encoding system.

2.6. Analysis of the fractal learning neural network's representation

We hypothesized that the FLNN would discover a fractal encoding like that of the hand wired dynamical automaton described in Section 1 above. Much of the interest of this hypothesis lies in the interpretation of the word "like." The analysis reported in Tabor (2009) suggests that even a minute distortion of a fractal grammar can result in a system whose behavior is formally (in terms of traditional computational classes) quite different from the undistorted version. Nevertheless, there is a clear sense in which a small divergence of the parameters from ideal values results in a system with structural features closely related to those of the ideal. Here, we define four tests that provide evidence that the trained FLNN model uses a structural system closely related to that of the ideal fractal grammar of Section 1. In the case of the FLNN, this correspondence is fairly self-evident in diagrams which we will show. One of our purposes in specifying these tests and showing how they work in the FLNN is to establish a foundation for a parallel analysis of the much less representationally transparent SRN, which we provide in the next section. Our other purpose is to point to ways in which the connectionist models in focus here appear to make different empirical claims and to be formally different from the models of classes 1 and 2 mentioned in the introduction.

The four tests are (a) Clustering of causal states, (b) Non-random structural self-similarity between and across scales, (c) Balanced contraction and expansion associated with pushes and pops, and (d) Ragged progressive generalization. We describe them each in turn.

2.6.1. Clustering of causal states—fractal learning neural network

Each state in the visitation set of DA1, the dynamical automaton presented above for L1, corresponds to a causal state of L1. We do not expect the trained FLNN to have a one-to-one correspondence between visited states and causal states because it is an inexact, learning model which converges on a solution only asymptotically. But if it is to employ approximately the same spatial fractal arrangement as DA1, then the causal states should correspond to non-overlapping clusters, and these clusters should be organized in the fractal pattern. Fig. 3 shows a sample visitation set from the trained model, with points belonging to the same causal state circled and triangles drawn between the means of the clusters under the same scheme as in Fig. 1. The figure was generated by generating a random sample of sentences from the training grammar (up to four concentric clauses), starting the linear hidden units in the state $\binom{0}{0}$ at the beginning of each sentence, and recording the visited states as the network processed the sentence. The z_1 and z_2 axes give the activations of the two linear hidden units. As the figure makes evident, the network has non-overlapping clusters corresponding to causal states, at least when tested on up to four concentric clauses.

2.6.2. Structural self-similarity—fractal learning neural network

Figure 3 makes it visually evident that the generalization is based on a fractal scheme. To explore this property objectively, we defined a measure of structural similarity across

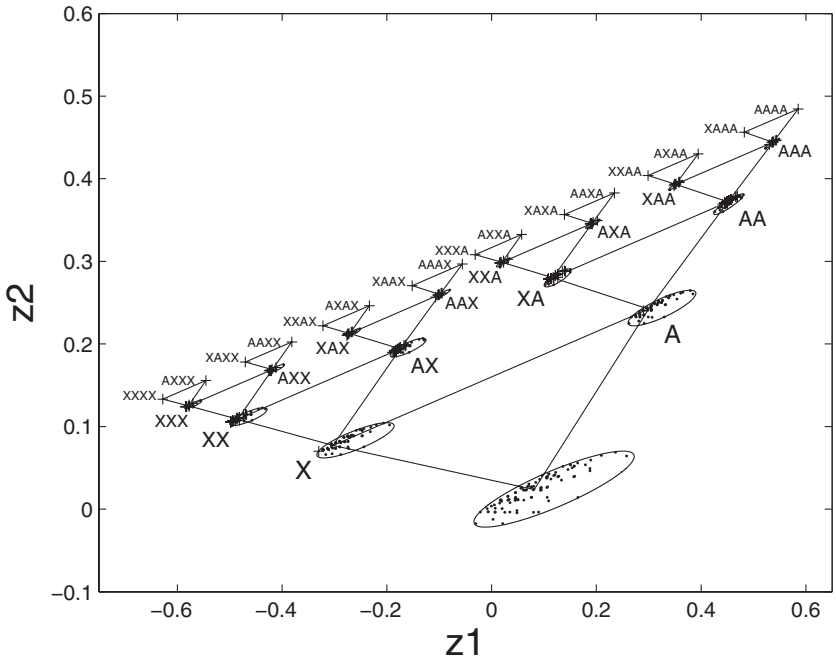

Fig. 3. The points visited during parsing of all sentences up to length 8 (maximum of four concentric levels). Points corresponding to the same PDA1 state are circled. Vertices of triangles are placed at the mean of the points in each circle, marked by crosses. The most deeply embedded causal states were only visited once each in the test sample, so no circles are drawn around these. See http://psychology.uconn.edu/labs/solab/papers.html#tabor2013csg for a color figure that distinguishes trained states from untrained.

the causal state means as follows. Two stack states are called *sisters* if they differ only in the bottom-most stack symbol (e.g., XAX and AAX are sisters—top of stack is on the right). Let $s_1 = AQ$ and $s_2 = XQ$, where Q is a finite sequence of stack symbols be stack states. We call the triangle formed by joining the means of the causal clusters corresponding to these stack states with each other and with the mean of the causal cluster corresponding to Q itself a *Local Triangle*. If the FLNN has developed a fractal scheme closely approximating the fractal grammar coding, then all possible local triangles should be *similar*: the ratios of corresponding side lengths should be the same. To measure the degree of this similarity, we consider all local triangles in a sample visitation set and define fractal misalignment (FM), *FM*, as follows:

$$FM = \frac{\sum_{i \in LT} \sum_{j<i, j \in LT} std\left(\frac{l_{Q(i)-AQ(i)}}{l_{Q(j)-AQ(j)}}, \frac{l_{Q(i)-XQ(i)}}{l_{Q(j)-XQ(j)}}, \frac{l_{AQ(i)-XQ(i)}}{l_{AQ(j)-XQ(j)}}\right)}{\frac{1}{2}|LT|(|LT|-1)} \tag{7}$$

Here, S(k) $(S \in Q, AQ, XQ, k \in i, j)$ is the stack state corresponding to a vertex of local triangle k, LT is the set of indices of all the local triangles derived from a sample visitation set; $|LT|$ is the cardinality of the set of indices; l_{M-N} is the length of the segment

joining the mean of the causal state with stack form *M* to the mean of the causal state with stack form *N*. If the triangles are all similar, as they are in DA1, the standard deviation, *std*, will be zero. To the extent that the triangles deviate from similarity, the measure will grow more positive. We note this measure captures some, but not all, of the structural regularity across the fractal (e.g., it does not detect differences in the orientations of the triangles with respect to one another).[8] For fractal grammars like DA1 given above, FM is identically zero. For the trained FLNN, the value is slightly positive but very close to zero if we examine the means of causal states with state resetting at the beginning of each sentence. In the section on the Simple Recurrent Network below, we show the measure FM provides helpful information about the SRN's structural development over the course of training.

2.6.3. Balanced contraction and expansion—fractal learning neural network

The two tests of the network encoding structure considered so far can yield positive outcomes in a connected space model that uses an infinite region to encode the state differences—for example, the causal clusters could, in principle be spread out over infinite area, and the local triangles could repeat their form across this infinite area. In some sense, this is what a context-free grammar or Turing Machine does using a stack or tape in which all cells make identical storage demands. The FLNN allows for this in principle, in that the range of its linear first hidden layer is unbounded. But a key feature of the fractal solution is that the infinity of the recursion is compressed into a bounded space via the balanced contraction and expansion of the push and pop events. This property amounts to a short-shrifting of the encoding resources used for more deeply embedded causal states relative to less deeply embedded causal states: If there is noise in the encodings, the noise distorts deeper embeddings more than shallow ones. This short-shrifting is plausibly related to the well-known limited ability of humans to process deep center-embeddings—see (Christiansen & Chater, 1999; Kirov & Frank, 2011). We would like, therefore, to objectively determine whether the system is exhibiting contraction for pushes and expansion for pops.

This is very easy in the case of the FLNN because the contraction and expansion are explicitly controlled by the hidden unit self-weights: The multiplier for "Push A" (i.e., the self-connection on each hidden unit when symbol a is presented on the input) should be less than 1 and the multiplier for "Pop A" (corresponding to the b symbol) should be greater than 1, and their product should be 1.[9] Likewise, the value for "Push X" should be less than 1 and the value for "Pop X" should be greater than 1. For the network at hand, we graphed these values over the course of training. All four values start at 1 (by design). After a several hundred epochs, the two push values decrease to a positive fraction and the two pop values increase symmetrically so they are roughly the inverse of the corresponding push values and the product stays near 1.[10] These findings indicate that the FLNN converges on a balanced contraction and expansion as in the fractal grammar model.[11]

2.6.4. Ragged progressive generalization—fractal learning neural network

To probe the structure of the networks' encoding over the course of its development, we fixed the weights at the end of each training epoch and examined the visitation set

associated with sentence-initial sequences of up to four pushes in a row (i.e., the locations reached at the ends of the strings a, x, aa, ax, xa, xx, aaa,… xxxx), always starting the network at the origin. If the network implements a fractal grammar, this set should be a complete inventory of the largest four scales of the fractal. Fig. 4 shows the development of this set over the course of learning.

Several properties of Fig. 4 are worth noting. The figure illustrates how a point can continuously deform into a totally disconnected fractal where the network of the final fractal is formally equivalent to (generates the same formal language as) a simple symbolic mechanism.[12] Using the target language as a reference point, the network becomes able, early in training, to process the sentences with the least embedding (ab, xy). But when the fractal branches are overlapping, the causal states are not all linearly separable from one another and the network makes mistakes on more deeply embedded sentences. Over the course of training, the branches gradually separate and increasingly many causal states are successfully processed. The figure suggests that the network extends its parsing ability with respect to the target language gradually, going far beyond its input, but succeeding in parsing some sentences at a given recursive level earlier than others. Fig. 5 shows success by individual sentence type as a function of training epoch for the sample network. There are ranges of epochs when the network seems to be making a transition from one level to another, but the transitions have a ragged character—some sentence forms succeed before others. It is important to note that this raggedness is not noise in the ordinary sense: At any one point in time, there is no noise in the weights or the activations of the network that is producing the inconsistent judgments between sentence forms at the same grammatical level. This suggests that the ragged behavior is part of the structure of the transitional network's representation, not the result of probabilistic sampling between two grammatical states, one of which rejects a sentence form and another of which accepts it. To emulate one of these intermediate stages with a symbolic model, we would need to write a rather complex symbolic description, perhaps one that simply had a separate rule for each sentence that the network judged grammatical. A simple form of ragged progressive generalization has been observed in all the previous studies of neural networks learning counting recursion: Depending on the random initial weights and the random variation in the training sequence, the network generalizes to different finite degrees of embedding beyond its training sample (e.g., Rodriguez, 2001). While symbolic models with variable stack capacity (Lewis, 1996) can show similar ragged behavior on linear state growth languages, this variable stack capacity approach fails to predict the ragged progressive pattern in exponential state growth languages.

The fractal unfolding illustrated in Fig. 4 is of interest because it allows us to examine emergence of form in a connectionist model at a level of description (computational) that gets at its general properties. In this sense, it helps us go beyond simulation methods. The result is also of interest with respect to the long-standing debate about whether language is a special human ability for which our genes tailor the outcome: Some connectionist networks are very general learning devices—they can be applied to a great range of problems across different domains and have fair success. The FLNN is not (at least not yet) known to be such a general device. There are several ways in which the system

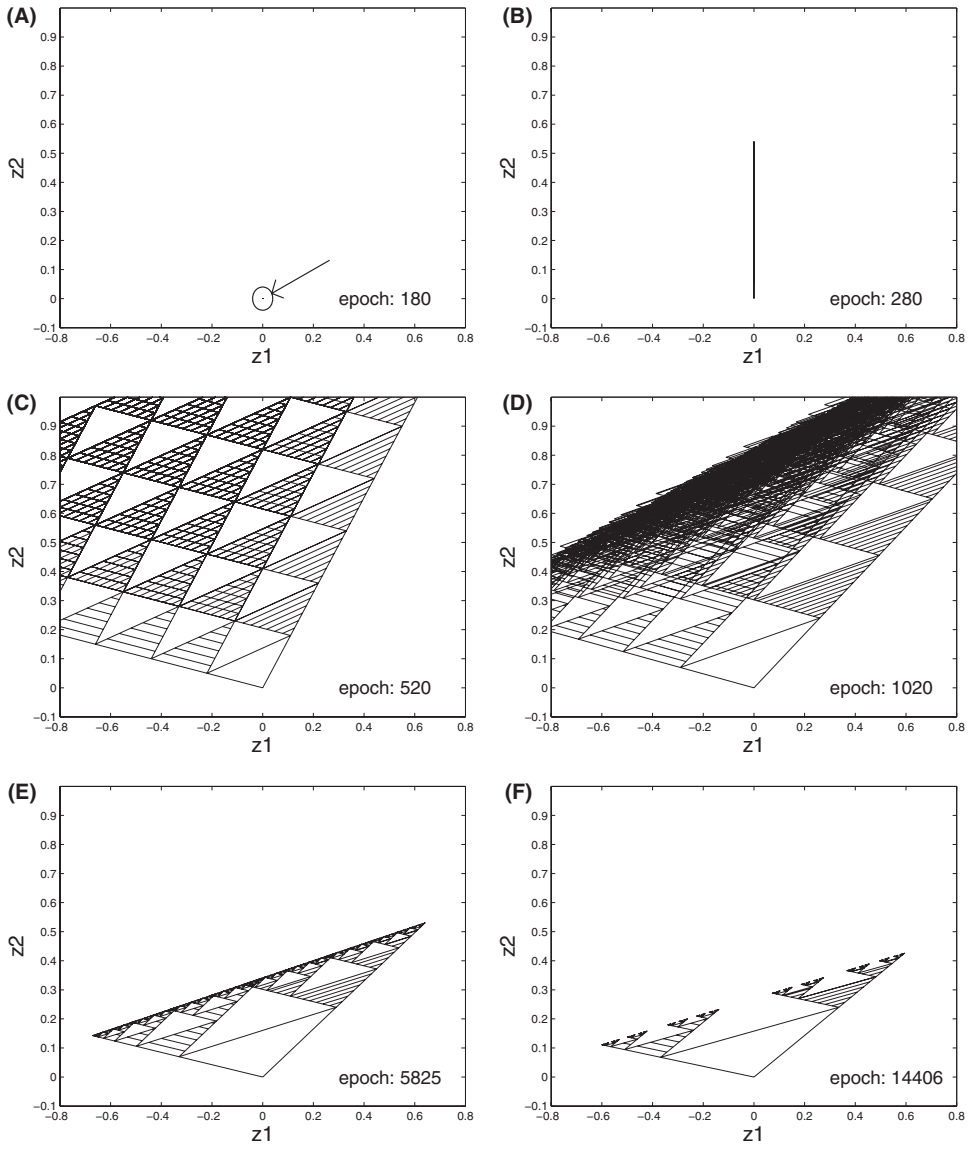

Fig. 4. The development of the fractal structure during learning of language L1. (A) At the beginning, the Linear Unit self-weights and the Input → Linear Unit weights are all zero, so inputs evoke no change in the hidden activations. (B) After some time, the visitation set lie on a one-dimensional manifold. In fact, the visited states lie at discrete points on this manifold, but these have been connected by a line segment in the figure to make them more visible. (C) The visitation set has expanded into two dimensions and is unbounded—each push causes a displacement of the same magnitude; in the figure, the visited points are connected by line segments and appear at the corners of triangles; the triangles have been shaded to distinguish states for which the top of stack is A (upper left to lower right shading) from states for which it is X (lower left to upper right shading). (D and E) The A and X branches of the set spread apart gradually and simultaneously shrink down to a small finite size; (F) eventually the set becomes a *totally disconnected fractal*—its branches do not overlap (Barnsley, [1988]1993).

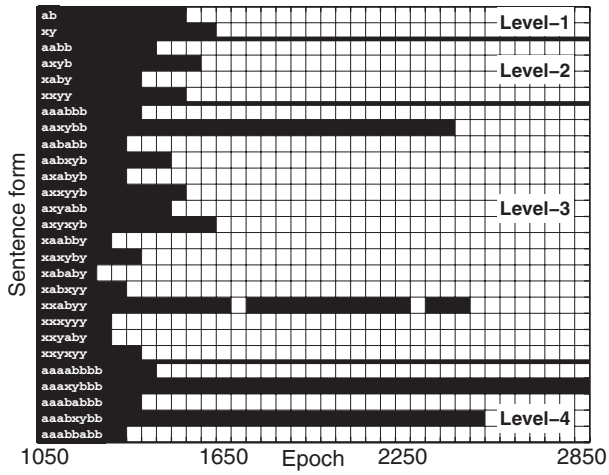

Fig. 5. Ragged progressive generalization in the FLNN. Each row corresponds to a sentence form. A black/white square indicates that parsing failed/succeed on every sentence of that form at the epoch corresponding to the column. The graph shows all of the Level 1, 2, and 3 sentences and a small sample of the many Level 4 sentences. For ease of viewing, only epochs 1,050–2,850 are displayed. Eventually (the network was trained to epoch 5,825) all but one Level 4 sentence was correctly parsed, indicating substantial generalization beyond the input. We call this "progressive generalization" because the network extends to higher levels of embedding on the basis of lower level training, and "ragged generalization" because not all sentences of the same level are successfully parsed at the same time.

is tailored to facilitate the outcome: The number of Linear Hidden Units and the number of Gaussian Units were chosen with knowledge of what is needed for the problem; the selective use of second-order connections and the absence of any non-self-connections between the Linear Hidden units restricts the search space to parameterizations that favor a fractal form; the self-weights on the two hidden units were yoked so that self-weights corresponding to the same input had to have the same value.[13] It is also possible that the very simple scheme for hill climbing, which only adjusts one weight at a time and employs only one step size, somehow narrows the search space so that the system is guided to the fractal form. For these reasons, it is worth asking whether principles of organization and learning suggested by this simulation extend to other connectionist learning regimes that are not so restricted. These considerations motivated an investigation of the much more general Simple Recurrent Network trained on a recursive language, which we report on in the next section.

3. Training a Simple Recurrent Network on a similar language

The SRN (Elman, 1990) is the most extensively studied neural network symbol processing model. It has made a range of plausible predictions about grammatical patterning (Elman, 1991), sentence processing (Christiansen & Chater, 1999; Rohde, 2002), and artificial grammar learning (Misyak et al., 2010) among other domains. As we indicated

above, it is also a more generic architecture than the FLNN. Furthermore, Elman (1991)'s results suggest that an SRN can learn recursive languages corresponding to pushdown automata with control state changes. Rodriguez (2001) trained an SRN on the palindrome language of Table 4. The network performed poorly—the best of 250 networks trained up to seven concentric levels only got 78% of the training sentences correct. Rodriguez found highly suggestive evidence of a fractal encoding scheme, using graphing and a hand-designed linear approximation (see Tabor, 2011). We now use the analysis methods developed for the FLNN above to probe a similar SRN, gleaning insight into how the network is like, but also unlike, a pushdown automaton.

We trained an SRN on a sequence generated by Language L2 (Table 4). We used L2, instead of L1 following Rodriguez, and also because we undertook parallel artificial grammar studies with humans (not reported here—see Cho et al., 2011; Tabor, Cho, & Szkudlarek, 2012), and we suspected that humans would have an easier time with L2 than L1: The average number of words over which stack symbols at a particular embedding level must be kept on the stack in L1 is longer than it is in L2; L2 has a period symbol, "p," which marks the end of every sentence; we think this makes it easier for humans to get oriented in the structure initially.

We implemented the core model with Michal Cernansky's code (http://www2.fiit.stuba.sk/~cernans/main/download.html). The activation and weight dynamics for this model are specified by Equations 8 and 9, respectively (Rumelhart, Hinton, & Williams, 1986).

$$\begin{array}{rcl} \mathbf{h}(\mathbf{t}) & = & f(W_{HH} \cdot \mathbf{h}(t-1) + W_{HI} \cdot \mathbf{s}(t) + \mathbf{b}_H) \\ \mathbf{o}(\mathbf{t}) & = & f(W_{OH} \cdot \mathbf{h}(t) + \mathbf{b}_O) \\ f(x) & = & \frac{1}{1+e^{-x}} \end{array} \tag{8}$$

$$\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}} \tag{9}$$

Here, $\mathbf{s}(t)$ is the vector of input unit activations at time step t, b_H and b_O are the biases on the hidden and output units, respectively, W_{HI} are the weights from input to hidden units, W_{HH} are the recurrent hidden connections, and W_{OH} are the weights from hidden to output. E is the total error across patterns, defined as summed Kullback-Leibler

Table 4
CFG2, the grammar of Language L2. This grammar specifies the probabilities of some productions with variables ($2p_1 + 2p_2 = 1$) because we did not use a stationary training distribution—see further discussion in the text

1.0	Root	→	S p
p_1	S	→	a S b
p_1	S	→	x S y
p_2	S	→	a b
p_2	S	→	x y

divergence between outputs and targets. The network had five input units (one for each word in L2), five output units, and ten hidden units.

3.1. Training

At each trial, the network received an indexical bit vector corresponding to one of the five sentence symbols on its input layer. It was trained on the task of predicting, on the output layer, which symbol would occur next at each point. The error gradient of (9) was approximated using Backpropagation Through Time (Rumelhart et al., 1986) with eight time steps unfolded. The constant of proportionality in (9) (called the "learning rate") was set to 0.05.

Following arguments by Newport (1990) that their limited working memories might help children learn complex languages by sparing them the trouble of dealing with complex temporal structures early on, Elman (1993) offered evidence that a SRN trained on recursive structure in batches (first Level 1, later Levels 1 and 2) did much better at learning the training corpus than one which received all sentence types from the start. However, Rohde and Plaut (1999) tried to replicate Elman's results and could not, arguing that "starting small" was not the issue. We hypothesized that graded training regimens are potentially helpful, but only if the staging is sensitive to the way a particular network (with particular random initial weights and a particular sequence of experience with the training data) is developing. We therefore trained 20 networks with different random initial weights progressively, exposing them first to Level 1 sentences, then to a mixture of Level 1s and 2s, and finally to a mixture of all Levels 1–3. The proportions of sentences of each form were determined by dividing sentence probability equally between levels (e.g., if Levels 1 and 2 were being trained, then 50% of the corpus were Level 1 sentences and 50% were Level 2) and dividing the probability among sentence forms equally within each level (e.g., there are four Level 2 sentences in L2, so for Level 1 and 2 training, the network saw each Level 2 sentence approximately 50/4 = 12.5% of the time). Rather than using a fixed regimen, we monitored the networks as they were learning and only presented the next recursive level after the network could correctly parse each current level.[14]

To check on the effectiveness of the adaptive training, we did a yoked training of 20 additional randomly initialized networks, assigning one of them to each of the original networks. The new network then received exactly the same sequence of training trials that its partner had received. If the adaptive training confers no advantage, we would expect no difference between the end of training accuracy for the original networks and the yoked networks.

3.2. Training results

To evaluate the networks' prediction accuracy at the end of training, we presented each of the 20 networks with a 1,400-sentence corpus generated randomly from the final-stage training distribution with learning turned off. We assessed the successful parsing of each

sentence in this sample by the same method as in the Training Results for the FLNN described above: The maximally activated output had to be a grammatical continuation, given the history of symbols observed, at every juncture. Unlike with the FLNN, though, we did not reset the hidden state at the start of every sentence. The average rate of correct sentence parsing of the adaptively trained networks was 98.4% (*SD* 0.69). We take this as evidence that the networks learned the structure of the training language well.[15]

For the 20 networks that did not receive adaptive training, the average end-of-training rate of correct parsing (evaluated with respect to the same 1,400 sentence test corpus) was 94.7% (*SD* 2.2). A paired *t*-test indicated the adaptively trained networks performed better than their yoked controls, suggesting that the progressive training confers an advantage.

3.3. Analysis of the Simple Recurrent Network's representation

We investigated structure in the hidden activation patterns of the SRN using the same four methods that we used to assess fractal structure in the Linear Hidden Units of the FLNN. We discuss each of these in turn.

3.3.1. Clustering of causal states—Simple Recurrent Network

We could not use simple graphing to assess the clustering of the causal states in the SRN's hidden space because it has 10 dimensions. Instead, we presented the network with the same 1,400 sentence testing corpus described in the preceding section and grouped the visitation set points by their causal states. We then asked, for each of these groups of points whether it was pairwise linearly separable from each other group. Testing for pairwise linear separability is a way of finding out if the causal states of the training environment correspond to clusters in the network's visitation set.

We tested linear separability between every unique pair of PDA states with a support vector machine (SVM; Boser, Guyon, & Vapnik, 1992; Cortes & Vapnik, 1995) employing a linear kernel function and the sequential minimal optimization algorithm (Platt, 1998) (see Elizondo, 2006 for review). An SVM learns to classify points in a vector space into two classes when presented with a set of training data and their corresponding classes. In our case, for each unique pair of PDA states, an SVM was trained to classify all their corresponding hidden state vectors into one of the two PDA states and then tested with the same data set. If the SVM correctly classified the hidden states into their corresponding PDA states with no exception, those two PDA states were judged to be linearly separable. The set of all hidden states that a network visited in the test run is pairwise linearly separable if every unique pair of PDA states is linearly separable. All 20 of the networks described passed this test for pairwise linear separability of causal states, consistent with our fractal model's prediction.

3.3.2. Structural self-similarity—Simple Recurrent Network

We expected the structural self-similarity of the visitation set to increase over the course of training, in keeping with our hypothesis that the SRN develops a fractal

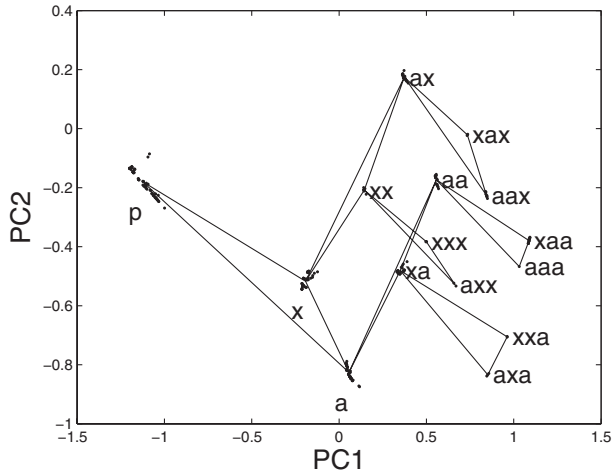

Fig. 6. A principal component projection of the hidden states of a sample trained SRN with visitation points for sequences of push trials shown. Triangles connect the means of causal states according to the scheme of Section *(2) Structural self-similarity—FLNN*.

encoding. Fig. 6 displays a principal component projection of the causal states of push sequences up to four concentric levels at the end of training of one sample network among the 20 trained (a similar figure can be constructed for pop states). Local triangles are drawn in the figure. To assess change in the self-similarity, we computed the FM of the network based on the causal state means derived in Section 1 *Clustering of causal states—SRN* above (sentences containing up to three concentric levels). For each of the 20 networks, we computed FM at the random initial state, and at the ends of training phases 1, 2, and 3. Almost all FM curves decreased monotonically over the course of training. A regression analysis indicated that the negative trend was significant, consistent with our hypothesis that the network should become more structurally self-similar over the course of training.[16]

3.3.3. Balanced contraction and expansion—Simple Recurrent Network

In the FLNN, we were able to examine the weights of the network directly to assess the claim that the pushes were contractive and the pops inverted the contraction. In the SRN, we do not know how to assess this property directly from the weights. Instead, we employed a language sample with up to four concentric levels (using the same distributional scheme as in the training corpus), examined the same types of triangles used in the three-point fractal misalignment analysis of the last section and recorded the ratios of sides of triangles across adjacent levels. For each of the 20 networks, we averaged all the side ratios associated with pairs of touching triangles, segregating the results by level and by whether a push or pop transition was involved. We computed these results at each stage of the network's training: random initial weights (phase 0) and the ends of phases 1, 2, and 3. Our prediction was that the SRN training would bring the push and pop

operations into a multiplicative inverse relationship, as we observed in the FLNN. Thus, we examined the product of the push ratio with the pop ratio over the course of training, expecting this value to approach 1. Unlike the FLNN, which was initialized at the ratio 1 and stayed close that value throughout training, the SRN started far from this ratio. Because, when the network has not converged on a perfect fractal, the ratio might either be greater or less than 1, we computed $|1 - ratio|$ and regressed this against phase. As predicted, the results showed a significant trend toward multiplicative balance between the pushes and pops.[17] We checked to make sure the final ratio corresponded to inversion rather than equality by considering average log of the push ratios and the average log of the pop ratios. The former was significantly below 0 (one-sample t-test, $p < .0001$) and the latter significantly above (one-sample t-test, $p < .0001$), in keeping with the fractal grammar prediction.

3.3.4. Ragged progressive generalization—Simple Recurrent Network

In the FLNN, we noted "ragged" generalization beyond the input on the way to very good approximation of the ideal recursive system. The SRNs did not generalize nearly as well—the networks fully trained on up to three concentric levels only got 5.8% (*SD* 7.5%) of the 963 test sentences right at four levels. Nevertheless, a subset of these networks showed robust evidence of generalization to sentences beyond their training sample at various times. At the end of the second phase of training, when the networks had not yet experienced any Level 3 sentences, six of the 20 trained networks showed persistent ability to correctly parse some but not all of the Level 3 sentences. Fig. 7 shows the

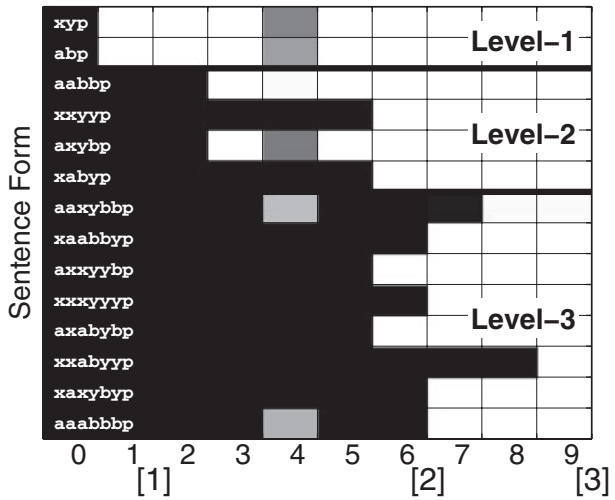

Fig. 7. Ragged progression and progressive generalization in an SRN. Each row corresponds to a sentence form. Each column corresponds to a training epoch. A black/white square indicates that parsing failed/succeed on every sentence of that form at the epoch corresponding to the column. Gray squares indicate intermediate rates of success. The numbers in square brackets at the bottom of the figure mark the ends of training phases: [N] = end of training on levels 1-N.

accuracy on all Level 1–3 sentences over the course of training for one of these generalizing SRNs. In this case, the network was tested on a sample corpus with up to Level 3 sentences. Unlike in the FLNN, where we tested each sentence from a fixed initial hidden state, the sentences in the SRN test were presented in the context of previous sentences and thus could vary in behavior across the test sample—this variable behavior is indicated by the gray shading in the figure. As in the FLNN case described above, the figure illustrates progressive generalization: Before the end of Phase 2 (training on Levels 1–2 only), the network exhibits correct parsing of some Level 3 sentences. This behavior is robust in the sense that under repeated tests, with different randomly sampled test corpora, the type and number of Level 3 sentences that are correctly parsed remains about the same. The figure also provides evidence for ragged generalization: Sentences from the same level start being correctly parsed at different times. It is notable that, in many cases of these ragged progressions, once a sentence starts being correctly parsed, it continues to be correctly parsed through the end of training. This suggests that the ragged developments bring the system toward the recursive solution: They are an aspect of the gradual emergence of the recursive structure. In the Conclusion, we point out some ways that this property might be used to clarify the relationship between the connectionist and probabilistic models mentioned in the Introduction.

4. Conclusions

We have presented evidence that a particular type of fractal encoding scheme arises when recurrent connectionist networks are trained to process center-embedding languages. We examined two kinds of recurrent networks: The FLNN, which is carefully designed to develop fractal systems in a conveniently linear recurrent layer, and the SRN, which has been explored in a much wider variety of settings and shows some robust learning characteristics but has a non-linear recurrent layer and is consequently harder to analyze. We identified four features that helped to diagnose fractal organization in both types of networks: (a) clustering of causal states, (b) structural self-similarity, (c) balancing of contraction and expansion, and (d) ragged progressive generalization. We also found that an adaptive training scheme, in which the SRN was presented with higher levels of embedding only after it had mastered lower levels, seemed to facilitate training.

As we suggested in the introduction, we think the main news is that it is possible to examine learned recurrent connectionist network representations at a level of description (the fractals) that is closely related that of symbolic mechanisms. This provides clarification of how the connectionist models work. It also offers a basis for further comparison between the approaches.

We return now to the point that motivated the article: explaining gradual aspects of language learning and language change and the relationship between approach [3]—the connectionist approach—and approaches [1] and [2], which explain gradualness via various kinds of probabilistic mixtures of symbolic behaviors. Our aim here is not to fully specify this relationship. We think that needs to be done via further mathematical and

empirical work. We suggest that the current observations provide helpful motivation and tools for that further work.

Regarding approach [1], which proposed that intermediate probabilities in sample data stem from mixtures of individuals, who themselves have simple symbolic grammars at all times, the current proposal makes a very simple suggestion: Seek evidence for gradualness within individuals. As a step in this direction, we have been exploring an artificial language learning paradigm called *Locus Prediction* in which participants learn complex recursive languages on the basis of exposure to data (Cho et al., 2011; Tabor et al., 2012). Crucially, the task of Locus Prediction provides word-by-word information about participant's expectations (see also Misyak et al., 2010) so the data can be compared in some detail to the predictions of the models described here.

Approaches [2a] and [2b], in the learning case, posit that the learner has a set of either grammars or rules at his or her disposal and adjusts probabilities on these to best fit the information provided by the training signal. To keep the system from overfitting the data, many such models posit an anti-complexity bias and a corresponding complexity-evaluation measure for symbolic systems. The anti-complexity bias, working in conjunction with a rational (e.g., Bayesian) inference mechanism generally predicts that the system will progress from simple models to more complex ones with greater complexity adopted only if it is warranted by the data (e.g., Perfors, Tenenbaum, & Regier, 2011; Xu & Tenenbaum, 2007). The connectionist models we have just reviewed show odd behavior in this regard: The models begin in what seems to be a very simple state (though it may contain hidden complexity in the random weights), and they ultimately approximate a somewhat more complex but still highly ordered behavior. On the way, they pass through states of behavior that seem to be very complex—odd subsets of the target language are accepted as grammatical. We recognize that we are examining the intermediate stages through the lens of the target language. It is possible that they would seem simpler if viewed from another point of view. However, they also may be very complex. Our work on related cases suggests that the intermediate states can be very complex (Tabor, 2009). This complexity stems from the interaction of the network's connected parameter space with the environment—in traveling continuously from one ordered state to another, the network passes through an uncountable infinity of intermediate parameter states. Some of these are associated with very complex behaviors. This suggests that two efforts will be helpful: One is to establish formally the nature of the structure of the intermediate states in the connectionist models. The other is to find out what people do in such cases. The current observations suggest that looking for highly complex intermediate behavior that is reliable (not noise) is a good way to distinguish between the approaches.

We raised the question in the Introduction whether the fact that Turing Machines can approximate connectionist devices arbitrarily closely means that the difference between the two approaches is so small that we might as well ignore it, especially from an empirical standpoint. Some questions need to be addressed before such a conclusion can be drawn. A natural approach to making Turing devices that approximate the connectedness of a particular connectionist device is to introduce a graph structure on a set of appropriately designed Turing Machines which recapitulates the adjacency relationships between

small neighborhoods in the connectionist parameter space. The idea is that changes of Turing Machine parameters are expected to proceed along paths of connected vertices in the graph, thus approximating continuous grammar change. But then the question arises, How does a language user who has just changed to a new Turing device decide how to parse the next word she encounters? In the connectionist model, the processing states before and after the parameter change all lie in the same space (e.g., the hidden unit space of the SRN), and change in the distribution over these states is continuous, so it works well to use the current state as an approximation of the new state. But for the Turing devices, there is not an obvious, principled set of proximity relationships between the states of one Turing Machine and the states of a different one—in many cases, the state spaces have different topologies. At this point, a natural move is to turn away from a focus on individual states, and toward a focus on the distribution over observed events under a probabilistic version of the machine (this is the approach taken by many computational linguistic methods which employ probabilistic rules and some measure of the fit between observed data and models with different rules—e.g., Perfors et al., 2011). But then the language user needs to consider a large sample of sentences to establish the current parse. Being obliged to parse the next word within a few hundred milliseconds, she cannot afford to collect a sample of millions of words before the decision is made. The only choice would seem to be to use the sample of previously observed utterances to select the parameters of the new grammar. If, however, we are trying to achieve fine-grained approximation of the changing grammars, this sample will not be very representative of the current language state because it includes many sentences generated by obsolete grammatical systems. Such a system would seem to have a bias toward the past. These considerations suggest that the distinction between the theories will be brought out by examining the advent of new forms—our intuition is that the Turing Machine induction approach will never change the rule system to generate structural types that have not previously been attested, as occurs in language change (Tabor, 1994, 1995) and therefore also in the transition between older generations and younger generations (in which language development may play a role).

Finally, we note a concept from dynamical systems theory that is helpful in further elucidating the nature of the connectionist systems (see also Kukona & Tabor, 2011; Tabor & Hutchins, 2004). A subset of dynamical systems exhibits a property called *self-organization*: Many small autonomously acting, but interacting, elements exhibit organized structure at the scale of the ensemble. Insect societies are a well-known case from ecology: Many interacting ants (or termites or bees) form a structure with global organizing principles even though a blueprint for this global organization does not appear to reside in any individual (Gordon, 2010). Self-organization arrives at order from disorder: Elements quite capable of failing to coordinate nevertheless coordinate due to continuous feedback interactions between them. We can find evidence for their self-organization by detecting systematicity in their non-conformity (Tabor, Galantucci, & Richardson, 2004). This is in contrast to an engineered system in which many parts are designed to behave in a precisely coordinated way and deviations have the form of noise. The neural networks that use fractals are more like the self-organizing systems; traditional symbolic

models are more like the engineered systems. In particular, where a symbolic model uses a single rule to specify behavior context independently in a large (infinite) sample of cases, the connectionist models use different parts of the fractal structure to handle the same range of cases. If the fractal is either perfect, or randomly fuzzy, then the connectionist model is indistinguishable from the symbolic models. But if there are deviations like those detected by the ragged progressive generalization analysis, then the self-organizing nature of the connectionist model becomes apparent: Many independent parts are coming together to give the appearance of a single rule. For present purposes, the interest of the self-organizing viewpoint is that the deviations, not being random, seem to be arrayed on complex pathways that lead from one ordered state to another. Discerning the structure of these pathways may help us understand how complex coordinated structure is able to come into existence in language learning and change.

Acknowledgments

We thank Harry Dankowicz and Daniel Müller-Feldmeth for insights about the network analysis and many invaluable discussions. We also thank three reviewers, one of whom was Paul Smolensky, whose extensive feedback helped us clarify key distinctions. We gratefully acknowledge support from NSF PAC grant 1059662 and NSF INSPIRE 1246920.

Notes

1. A space X is *connected* if it cannot be divided into two disjoint, non-empty open sets (Munkres, 2000). For example, R and intervals between pairs of points in R are connected spaces under the standard "order topology" on R, based on the ordering of the real numbers. Sets of discrete points in R, like the integers and rationals, are not connected. Countable sets naturally have a discrete topology, making them disconnected.
2. One can create a connected topology for a countable set, but there is not a single, obvious way of doing so, and there may be some challenges to doing so in the case of Turing Machines—see Conclusions.
3. A formal language is called a *counting recursion* language if it can be processed by a pushdown automaton with only one stack symbol—that is, the stack only serves as a counter; it need not keep track of embedding order. In counting recursion languages, the number of stack states needed for processing grows linearly with the length of the string. Such *linear state growth* languages may be contrasted with *exponential state growth* languages, where the number of stack states grows exponentially with the length of the string. *Mirror recursion languages*, in which the stack needs more than one symbol and grows and shrinks exactly once per sentence, are a type of exponential state growth language.

4. A *metric space* S is a space in which a distance metric is defined. A *distance metric* is a function d: $S \times S \rightarrow R$ satisfying $d(x,x) = 0$, $d(x,y) = d(y,x)$ and $d(x,y) + d(y,z) \geq d(x,z)$ for $x,y,z \in S$. Introducing a metric, here, is a way of taken advantage of the connectedness assumed in the introduction to make precise the notion, "arbitrarily small changes" mentioned there. A space X is *complete* if it contains all of its limit points. A point $s \in S$ is called a *limit point* of $X \subseteq S$ if every neighborhood of s contains an element of X that is different from s. Completeness is helpful because some of the limit points give rise to super-Turing behaviors that are not otherwise exhibited. This helps clarify the difference between symbolic and connected space "computation" (see Tabor, 2009).
5. Some of the languages generated by the connected space computers are not computable (a.k.a. "Super-Turing"). The non-computable languages are not sequestered in obscure corners of the parameter spaces but are densely interleaved with the computable ones. Tabor (2009) therefore suggests moving away from a perspective whose overriding concern is to pin down the presumed single nature of each computational process and toward a perspective in which processes are viewed in proximity and path relationships to one another.
6. Having introduced probabilities of rules, we will now assume that the sentences generated by the grammar are repeatedly sampled and strung end to end forever. We will thus switch to using the term *language* to refer to an assignment of probabilities to branches in the infinitely branching dendogram which specifies all the possible one-sided infinite strings that can be generated with the symbols of the alphabet.
7. In the language's dendogram, nodes correspond to transitions between symbols. We will consider two nodes to be identical if the subtrees they beget are exactly the same—same branching structure and same probabilities forever. The causal states are then the equivalence classes of nodes.
8. The measure can be straightforwardly extended to probe fractal organization more fully by having it assess the similarity of more complex polygonal simplexes.
9. Recall that the recurrent connections were yoked between the hidden units so these values are the same on both hidden units.
10. When the first divergence from 1 occurs, it occurs simultaneously for both push and pop, but contraction outweighs the expansion briefly, so the product becomes less than one and then it migrates back slowly toward 1 over the course of the remaining training.
11. An alternative method of testing for balance examines Lyapunov exponents—these give the average of the log of the rate of contraction in all principal directions on the attractor of an autonomous dynamical system; adopting a generalization suitable to the type of driven systems considered here, the largest Lyapunov exponent is 0 in a fractal grammar driven by a grammatical sample, and our earlier work (Tabor, 2002) suggests that it tends to zero in SRNs trained on stack and queue languages. Because our interest here is in understanding the explicit relationship between the fractal model and the actions of a PDA, we examined contraction and expansion separately.

12. In fact, the trained network is not exactly equivalent to PDA1, because the tuning of the pop and push relationship is not perfect, but it is very close (see the previous section) and it is easy to understand conceptually how it could be perfect.
13. The model still discovers fractal solutions without this constraint, but the solutions are generally closer to an idealized fractal when this constraint is enforced.
14. The network was tested after every 200 sentence presentations as follows: For each unique sentence form in the training corpus at a given phase, we initialized the hidden units to the state they occupied at the end of the last training sentence. We then presented three Level 1 sentences in a row to bring the hidden units to an appropriate sentence-initial state, then presented the test sentence, checking the pop transitions, which have unique next symbols, to see whether the unit corresponding to the correct next symbol was the highest activated among the five output units. If the network passed this test for every distinct sentence form at a given level, training on the next level commenced.
15. Although these results compare favorably to the considerably lower figures reported by Rodriguez (2001), we only trained our networks up to three concentric levels, compared to his seven. We stopped at three because our purpose was not to achieve high accuracy at great depth but to examine the principles of the encoding scheme.
16. The empirical FMs were modeled separately, using a linear mixed-effects model (Baayen, Davidson, & Bates, 2008). We ran the analysis of FM twice, once for states following pushes and the other for states following pops, in R, using the lme4 package. In both cases, the dependent variable was the empirical FM and the fixed-effect variable was training phase number ("PhaseNo"). The NetID was included for random intercepts. The effect of PhaseNo was evaluated by comparing a model with PhaseNo as a predictor and one without it, using a likelihood ratio test. The effect of PhaseNo was significant both for push FMs ($b = -0.481$, $b_{se} = 0.083$, $x^2(1) = 26.89$, $p < .0001$) and for pop-FMs ($b = -0.242$, $b_{se} = 0.021$, $x^2(1) = 79.00$, $p < .0001$).
17. We used a linear mixed-effects model (Baayen et al., 2008). As in the FM analysis, the models were fitted in R, using the lme4 package. The dependent variable was the distance from perfect balance; the fixed-effect variables were a continuous variable, PhaseNo, and a categorical variable, Level. The random intercepts of NetID were included to control the interdependency among the data points. The fixed effects were evaluated by model comparison, adding a fixed-effects term while keeping the same random-effects term. The base model (Model0) had only the intercept term. Adding PhaseNo to the base model (Model1) decreased the deviance measure significantly ($x^2(1) = 254.18$, $p < .0001$). Adding Level to this new model (Model 2) decreased the deviance measure again ($x^2(1) = 46.83$, $p < .0001$). However, adding the interaction term between two predictors (Model3) did not decrease the deviance measures further ($x^2(1) = 0.32$, $p = .570$). The coefficients of PhaseNo and Level from Model 2 were -0.280 (SE = 0.010) and 0.160 (SE = 0.022), respectively. The positive coefficient of Level in Model

2 indicates the ratio was more distant from perfect balance when the ratio was based on Level 2 versus 3, as compared to when it was based on Level 1 versus 2. This may be due to the fact that the network was only trained up to Level 3 and only partially generalized to Level 4, which is involved in the computation of the Level 2 versus 3 ratio on the pop side. The negative coefficient of PhaseNo in Model 2 is consistent with our prediction that training drives the ratio to 1.

References

Altmann, G., & Kamide, Y. (1999). Incremental interpretation at verbs: Restricting the domain of subsequent reference. *Cognition*, *73*, 247–264.

Andersen, H. (1987). From auxiliary to desinence. In M. Harris & P. Ramat (Eds.), *Historical development of auxiliaries* (pp. 21–52). Berlin: Mouton de Gruyter.

Baayen, R. H., Davidson, D. J., & Bates, D. M. (2008). Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language*, *59*(4), 390–412.

Barnsley, M. ([1988]1993). *Fractals everywhere*. (2nd ed.) Boston, MA: Academic Press.

Blair, A. D., & Pollack, J. (1997). Analysis of dynamical recognizers. *Neural Computation*, *9*(5), 1127–1142.

Blum, L., Cucker, F., Shub, M., & Smale, S. (1998). *Complexity and real computation*. New York: Springer-Verlag.

Blum, L., Shub, M., & Smale, S. (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.

Bodén, M., & Blair, A. (2003). Learning the dynamics of embedded clauses. *Applied Intelligence*, *19*, 51–63.

Bodén, M., & Wiles, J. (2002). On learning context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, *13*(2), 491–493.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In D. Haussler (Ed.), *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* (pp. 144–152). New York: ACM New York.

Bryant, V. (1985). *Metric spaces, iteration and application*. Cambridge, UK: Cambridge University Press.

Calvert, W. (2011). Metric structures and probabilistic computation. *Theoretical Computer Science*, *412*, 2766–2755.

Chalup, S., & Blair, A. D. (2003). Incremental training of first order recurrent neural networks to predict a context-sensitive language. *Neural Networks*, *16*, 955–972.

Cho, P. W., Szudlarek, E., Kukona, A., & Tabor, W. (2011). An artificial grammar investigation into the mental encoding of syntactic structure. In L. Carlson, C. Hoelscherr, & T. F. Shipley (Eds.), *Proceedings of the 33rd Annual Meeting of the Cognitive Science Society (cogsci2011)* (pp. 1679–1684). Austin, TX: Cognitive Science Society.

Christiansen, M. H., & Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, *23*, 157–205.

Corballis, M. C. (2007). Recursion, language, and starlings. *Cognitive Science*, *31*, 697–704.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Craig, C. (1991). Ways to go in Rama: a case study in polygrammaticalization. In E. C. Traugott & B. Heine (Eds.), *Approaches to grammaticalization, v. 2* (pp. 455–492). Amsterdam, The Netherlands: John Benjamins.

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).

Downey, R. G., & Hirschfeldt, D. R. (2010). *Algorithmic randomness and complexity*. New York: Springer-Verlag.

Elizondo, D. (2006). The linear separability problem: Some testing methods. *Neural Networks, IEEE Transactions on*, *17*(2), 330–344.

Ellegård, A. (1953). *The auxiliary do*. Stockholm, UK: Almqvist & Wiksell.

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.

Elman, J. L. (1993). Learning and development in neural networks: the importance of starting small. *Cognition*, *48*, 71–99.

Fontana, J. M. (1993). *Phrase structure and the syntax of clitics in the history of Spanish* (Unpublished doctoral dissertation). University of Pennsylvania, Philadelphia, PA.

Gordon, D. M. (2010). *An encounters: Interaction networks and colony behavior*. Princeton, NJ: Princeton University Press.

Haken, H. (2004). *Synergetic computers and cognition, 2nd enlarged edition*. Berlin: Springer.

Hare, M., & Elman, J. L. (1995). Learning and morphological change. *Cognition*, *56*, 61–98.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages, and computation*. Menlo Park, CA: Addison-Wesley.

Kirov, C., & Frank, R. (2011). Processing of nested and cross-serial dependencies: An automaton perspective on srn behavior. *Connection Science*, *24*, 1–24.

Kolen, J. F., & Kremer, S. C. (Eds.). (2001). *Field guide to dynamical recurrent networks*. New York: IEEE Press.

Kroch, A. S. (1989a). Function and grammar in the history of English: periphrastic do. In R. W. Fasold & D. Schiffrin (Eds.), *Language change and variation* (pp. 134–169). Philadelphia: John Benjamins. (Published as Vol. 52 of the series Current Issues in Linguistic Theory.)

Kroch, A. S. (1989b). Reflexes of grammar in patterns of language change. *Journal of Language Variation and Change*, *1*(3), 199–244.

Kukona, A., & Tabor, W. (2011). Impulse processing: A dynamical systems model of incremental eye movements in the visual world paradigm. *Cognitive Science*, *35*, 1009–1051.

Labov, W. (1972). *Language in the inner city: Studies in the Black English vernacular*. Philadelphia: University of Pennsylvania Press.

Lewis, R. (1996). Interference in short-term memory: The magical number two (or three) in sentence processing. *Journal of Psycholinguistic Research*, *25*(1), 93–115.

Magnuson, J. S., Tanenhaus, M. K., Aslin, R. N., & Dahan, D. (2003). The time course of spoken word learning and recognition: Studies with artificial lexicons. *Journal of Experimental Psychology*, *132*(2), 202–227.

Minsky, M., & Papert, J. (1988[1969]). *Perceptrons: An introduction to computational geometry*. Cambridge, MA: MIT Press.

Misyak, J. B., Christiansen, M. H., & Tomblin, J. B. (2010). Sequential expectations: The role of prediction-based learning in language. *Topics in Cognitive Science*, *1*, 138–153.

Moore, C. (1996). Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, *162*(1), 23–44.

Moore, C. (1998). Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, *201*, 99–136.

Munkres, J. R. (2000). *Topology (2nd Ed).* Upper Saddle River, NJ: Prentice Hall.

Newport, E. (1990). Maturational constraints on language learning. *Cognitive Science*, *14*(1), 11–28.

Nies, A. (2009). *Computability and randomness*. Oxford, UK: Clarendon Press.

O'Grady, W. (1997). Syntactic development. Chicago, IL: University of Chicago Press.

Perfors, A., Tenenbaum, J., & Regier, T. (2011). The learnability of abstract syntactic principles. *Cognition*, *118*, 306–338.

Platt, J. C. (1998). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in Kernel methods: Support vector learning* (pp. 185–208). Cambridge, MA: MIT Press.

Plunkett, K., & Marchman, V. (1991). U-shaped learning and frequency effects in a multi-layer perceptron: Implications for child language acquisition. *Cognition*, *38*, 43–102.

Pollack, J. (1987). *On connectionist models of natural language processing* (Unpublished doctoral dissertation). University of Illinois, Urbana, IL.

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

Rodriguez, P. (2001). Simple Recurrent Networks learn context-free and context-sensitive languages by counting. *Neural Computation*, *13*(9), 2093–2118.

Rodriguez, P., Wiles, J., & Elman, J. (1999). A recurrent neural network that learns to count. *Connection Science*, *11*(1), 5–40.

Rohde, D. (2002). *A connectionist model of sentence comprehension and production* (Doctoral dissertation, Carnegie Mellon University). Retrieved from http://tedlab.mit.edu/ ~ dr/Thesis/. Accessed June 4, 2013.

Rohde, D., & Plaut, D. (1999). Language acquisition in the absence of explicit negative evidence: How important is starting small? *Journal of Memory and Language, 72*, 67–109.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group (Eds.), *Parallel distributed processing, v. 1* (pp. 318–362). Cambridge, MA: MIT Press.

Rumelhart, D. E., & McClelland, J. L. (1986). On learning the past tenses of English verbs. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition (Vol. 1)* (pp. 216–271). Cambridge, MA: MIT Press.

Servan-Schreiber, D., Cleeremans, A., & McClelland, J. L. (1991). Graded state machines: The representation of temporal contingencies in Simple Recurrent Networks. *Machine Learning*, *7*, 161–193.

Siegelmann, H. T. (1999). *Neural networks and analog computation: Beyond the Turing limit*. Boston, MA: Birkhäuser.

Siegelmann, H. T., & Sontag, E. D. (1991). Turing computability with neural nets. *Applied Mathematics Letters*, *4*(6), 77–80.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences*, *11*(1), 1–74.

Smolensky, P. (1999). Grammar-based connectionist approaches to language. *Cognitive Science*, *23*, 589–613.

Tabor, W. (1994). *Syntactic innovation: A connectionist model* (Doctoral dissertation, Stanford University). Retrieved from http://psychology.uconn.edu/labs/solab/papers.html. Accessed June 4, 2013.

Tabor, W. (1995). Lexical change as nonlinear interpolation. In J. D. Moore & J. F. Lehman (Eds.), *Proceedings of the 17th Annual Cognitive Science Conference* (pp. 242–247.) Hillsdale, NJ: Lawrence Erlbaum Associates.

Tabor, W. (2000). Fractal encoding of context-free grammars in connectionist networks. *Expert Systems: The International Journal of Knowledge Engineering and Neural Networks*, *17*(1), 41–56.

Tabor, W. (2002). The value of symbolic computation. *Ecological Psychology*, *14*(1/2), 21–52.

Tabor, W. (2003). Learning exponential state growth languages by hill climbing. *IEEE Transactions on Neural Networks*, *14*(2), 444–446.

Tabor, W. (2009). A dynamical systems perspective on the relationship between symbolic and non-symbolic computation. *Cognitive Neurodynamics*, *3*(4), 415–427.

Tabor, W. (2011). Recursion and recursion-like structure in ensembles of neural elements. In H. Sayama, A. Minai, D. Braha, & Y. Bar-Yam (Eds.), *Unifying Themes in Complex Systems. Proceedings of the VIII International Conference on Complex Systems*, (pp. 1494–1508). Cambridge, MA: New England Complex Systems Institute, Available at: http//necsi.edu/events/iccs2011/proceedings.html. Accessed June 4, 2013.

Tabor, W., Cho, P. W., & Szkudlarek, E. (2012). Fractal unfolding: A metamorphic approach to learning to parse recursive structure. In D. Reitter & R. Levy (Eds.), *Proceedings of the 3rd Workshop on Cognitive*

Modeling and Computational Linguistics (CMCL 2012) (pp. 21–30). Montréal, Canada: Association for Computational Linguistics.

Tabor, W., Galantucci, B., & Richardson, D. (2004). Effects of merely local syntactic coherence on sentence processing. *Journal of Memory and Language*, *50*(4), 355–370.

Tabor, W., & Hutchins, S. (2004). Evidence for self-organized sentence processing: Digging in effects. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *30*(2), 431–450.

Tabor, W., Juliano, C., & Tanenhaus, M. (1997). Parsing in a dynamical system: An attractor-based account of the interaction of lexical and structural constraints in sentence processing. *Language and Cognitive Processes*, *12*(2/3), 211–271.

Tanenhaus, M., Spivey-Knowlton, M., Eberhard, K., & Sedivy, J. (1995). Integration of visual and linguistic information during spoken language comprehension. *Science*, *268*, 1632–1634.

Wells, C. G. (1985). *Language development in the pre-school years*. (Language at home and at school, Volume 2). New York: Cambridge University Press.

Wiles, J., & Elman, J. (1995). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks. In J. D. Moore & J. F. Lehman (Eds.), *Proceedings of the 17th Annual Cognitive Science Conference* (pp. 482–487). Hillsdale, NJ: Lawrence Erlbaum Associates.

Xu, F., & Tenenbaum, J. B. (2007). Word learning as Bayesian inference. *Psychological Review*, *114*(2), 245–272.