Tabor, W. (2011). Recursion and Recursion-Like Structure in Ensembles of Neural Elements. Sayama, H, Minai, A., Braha, D., & Bar-Yam, Y. (eds.) Unifying Themes in Complex Systems. Proceedings of the VIII International Conference on Complex Systems. pp. 1494-1508.
http://necsi.edu/events/iccs2011/proceedings.html

Recursion and Recursion-Like Structure in Ensembles of Neural Elements

Whitney Tabor
University of Connecticut
whitney.tabor@uconn.edu

Connectionist models have used general principles to model phenomena across many mental domains. Thus, they seem promising for uniting diverse syntactic phenomena (e.g., language, music, action). A challenge has been understanding how recursion can work in neurons. Headway has been made in training Recurrent Neural Networks (RNNs) to process languages which can be handled by the use of one or more symbol-counting stacks (e.g., $a^n b^n$, $a^n A^m B^m b^n$, $a^n b^n c^n$). Success with *exponential state growth languages*, in which stacks function as full-fledged sequence memories (e.g., palindrome languages, certain natural language relative clause constructions), has not been as great. [24] introduces Fractal Learning Neural Networks (FLNNS), showing that they can learn some exponential state growth languages with high accuracy. The current paper analyzes the performance of these FLNNs clarifying the relationship between their imperfect, but nevertheless structurally insightful, neural recursive encoding, and the perfect recursive encodings of symbolic devices.

1 Introduction

Recurrent neural networks (RNNs) of sufficient size can implement Turing machines [14, 21] and thus perform the same computations as symbolic computing mechanisms. It has been difficult, however, to successfully train RNNs to learn arbitrary infinite-state languages. This paper focuses on RNN learning of context-free languages (CFLs), the smallest class of infinite state languages on the Chomsky Hierarchy [3], and the class that forms the foundation of many formal theories of natural language syntax. (Online Appendix

1—http://solab.uconn.edu/People/Tabor/papers.html—provides definitions of terms.) An early effort to let a sequential neural symbol processor induce a context free language used a neural controller to manipulate a separate neural stack [22]. More recently, several projects have been able to induce the stack mechanism itself, as well as the controller, in the network's hidden units. Successful results have been obtained for languages that use one or more stacks as "counters". For example, a Simple Recurrent Network (SRN, [4]) trained by [27] correctly learned the corpus of sentences from the language $a^n b^n$ with up to 11 levels of embedding and generalized up to 18 levels. [2] found that the Sequential Cascaded Network (SCN) of [15] achieved generalization considerably beyond the training set on $a^n b^n c^n$, a context-sensitive language.

Because infinite-state languages involve arbitrarily long temporal dependencies, these projects have struggled with the well-known exponential decay of the error signal over time in standard RNN training regimens—see [8]. Therefore, [9] developed Long Short-Term Memory (LSTM) networks, which employ a special kind of unit that maintains constant error flow across arbitrarily long dependencies. Indeed [6] found that LSTM networks trained on examples from $a^n b^n$ and $a^n b^n c^n$ with at most 10s of levels of embeddings generalized perfectly to 1000s of levels (see [24]). LSTM networks also did well with the "restricted palindrome language", $a^n A^m B^m b^n$, far exceeding the performance of previous RNNs.

Whereas formal automata are naturally suited to modeling formal symbolic languages, where what matters is what occurs, not how often it occurs, learning neural networks are sensitive to statistical properties of their training data. Since symbol-processing RNNs combine these viewpoints, it is helpful to establish terminology for discussing probabilities of symbol sequences.

A *corpus-distribution* is a map from each prefix (i.e., sequence of words from the vocabulary) to a probability distribution over next-words. The *probability of a word sequence* is the product of the probabilities of the successive word-to-word transitions within it. A word sequence is a *generable string* of a corpus-distribution if it has nonzero probability. A machine *correctly processes* a generable string from a corpus distribution if, upon being presented with each symbol of the string in succession, it accurately specifies the probabilities of the symbols that follow. An activation vector *accurately specifies* a probability distribution if it is closer to the correct distribution than to any other distribution associated with a symbol transition in the corpus. This method of assessing correctness is equivalent to the method used by a number of neural net researchers in cases where only one next-word is possible: count the prediction as correct if the activation of the unit associated with the correct next word is above 0.5 [17, 2, 18]. The current method has the advantage of also being useful in cases where non-absolute probabilities are involved.

The languages mentioned so far constitute a proper subset of stack-manipulation languages. In particular, all of them can be modeled with a device that counts stack symbols on one or more stacks [6, 17]; none require keeping track of arbitrary orders of the symbols on the stack. For example, to process $a^n b^n c^n$, one stack can count up as the a's occur and down as the b's occur; a

second stack can count up as the b's occur and down as the c's occur. There are many stack-manipulation languages, even within the set of context-free languages, which require keeping track of the order, as well as the number, of stack elements. For example each sentence in wW^R (a palindrome language) consists of strings that can be divided in half such that each half is a mirror-image (under homomorphism) of the other half—e.g. "a b a a A A B A".

The *state growth function* of a language L is the function which specifies, for each possible sentence-length, n, how many states a computer must distinguish in order to correctly process all strings of length $\leq n$ from L. Infinite-state languages which require keeping track of arbitrary orders of elements on their stacks (like wW^R) are *exponential state-growth languages*. [24] provides evidence that most prior efforts at learning infinite state languages with RNNs have achieved high accuracy with linear or quadratic state growth languages, but not with exponential state growth languages.

2 Dynamical Automata and Fractal Learning Neural Networks

Putting aside the problem of *learning* exponential state growth languages, [12], [21], [23], [1], and [25] show that fractal sets provide a natural framework for encoding recursive computation in analog computing devices.

2.1 Pushdown Dynamical Automata for Context Free Languages

[23] defines *dynamical automata* for processing symbol sequences in a bounded real-valued activation space. A *dynamical automaton* (or "DA") is a symbol processor whose states lie in a complete metric space. The DA must start at a particular point called the *start state*. It is associated with a list of state change functions that map the metric space to itself, a vocabulary of symbols that it processes, and a partition of the metric space. An *Input Map* specifies, for each compartment of the partition, a set of symbols that can be grammatically processed when the network is in the current partition, and the state change function that is associated with each symbol. Online Appendix 2 provides a formal definition.

[23] defines a subset of Dynamical Automata, called *Pushdown Dynamical Automata* (PDDAs), that behave like Pushdown Automata (PDAs—see online Appendix 1). PDDAs travel around on fractals with several branches. Their functions implement analogs of stack operations: pushing, popping, and exchanging symbols. Pushing involves moving to a part of the fractal where the local scale is an order of magnitude smaller than the current scale; popping is the inverse of pushing; and exchanging involves shifting position at the largest scale without changing the scale. [23] shows that the set of languages processed

Table 1: The Input Map for Pushdown Dynamical Automaton 1 (PDDA 1). The initial state is $\binom{0}{0}$. When the automaton is in the state listed under "Compartment", it can read (or emit) the symbol listed under "Input". The consequent state change is listed under "State Change".

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

Table 2: Grammar 1. Parentheses denote optional constituents, which occur with probability 0.2 in every case. A probability, indicated by a decimal number, is associated with each production. The probabilities are irrelevant for implementation of a dynamical automaton but are useful for generating a corpus which can be used to train a FLNN.

1.0 S → A B C 1.0 A → a (S) 1.0 B → b (S) 1.0 C → c (S)

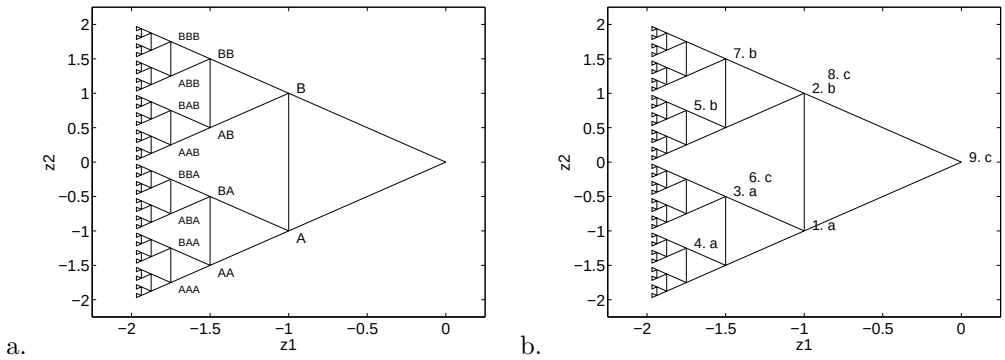
by PDDAs is identical to the set of languages processed by PDAs, that is to say, the Context Free Languages [11].

Table 1 gives an example of a PDDA for processing the sentences generated by the grammar shown in Table 2. PDDA 1 always starts at the origin of the space $z_1 \times z_2$ at the beginning of processing a sentence; it must move in accordance with the rules specified in Table 1; if it is able to process each word of the sentence and return to the origin when the last word is read, then the sentence belongs to the language.

In order to successfully process Language 1 with a symbolic stack, it is necessary to store information on the stack whenever an "a b c" sequence is begun but not completed. If **A** is stored when "a" is encountered, and **B** is stored when "b" is encountered, then the possible stack states are the members of {**A**, **B**}*. As Figure 1a shows, PDDA 1 induces a map from this set to positions on a fractal with the topology of a Sierpinski Gasket [1]. Figure 1b provides an example of the states visited by the automaton as it processes the center-embedded sentence, "a b a a b c b c c".

2.2 Insight from Prior Training Results

[17] trained a SRN on several context free languages. Although the networks did not achieve high accuracy on exponential state-growth languages, [17] was able to examine the network weights and construct effective solutions, using a linear recurrent map in place of the SRN's sigmoidal hidden unit activation function. For example, based on a network trained on wW^R, [17] defined the hidden-layer



a. b.

Figure 1: a. The stack map of PDDA 1. 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 stack is the right-most letter in each label. b. The trajectory associated with the sentence, "a b a a b c b c c", from Language 1. "1." identifies the first word, "2." the second, etc.

map shown in (1)-(2).

$$\vec{z}_t = f\left(\begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \end{bmatrix} \cdot \vec{z}_{t-1} + \begin{bmatrix} 0.5 & 0.5 & -5 & -5 \\ 0.4 & 0.1 & -5 & -5 \\ -5 & -5 & -1 & -1 \\ -5 & -5 & -0.8 & -0.2 \end{bmatrix} \cdot \vec{I}_t\right) \quad (1)$$

$$f_i(n_i) = \left\{ \begin{array}{lll} 0, & & n_i < 0 \\ z_i, & 0 \leq & n_i \leq 1 \\ 1, & 1 < & n_i \end{array} \right\} \quad (2)$$

The input, I_t, is encoded as a = (1, 0, 0, 0); b = (0, 1, 0, 0); A = (0, 0, 1, 0); B = (0, 0, 0, 1). For the "push" operations associated with the symbols "a" and "b", the third and fourth dimensions have fixed values, so it is possible to graph the important state changes. Figure 2a shows the set of all states visited by the map as it processes all possible initial sequences of a's and b's down to seven levels of embedding (the "push set").

Figure 2a makes it clear that Rodriguez's gleaned model uses the same kind of stack mechanism as a PDDA. Two insights from this and other prior work help with the challenge of learning exponential state-growth languages: (i) as [17] and [10] point out, the fractal scaling solution depends critically on being able to invert the scale changes. SRNs do this by using the linear region of their activation function to approximate multiplicative inverses. But the imperfection of the approximation leads to inaccuracies at high levels of embedding—hence the use by [17] of the piecewise linear function, f. The FLNNs discussed in the next section avoid the nonlinear distortion by using linear recurrent units

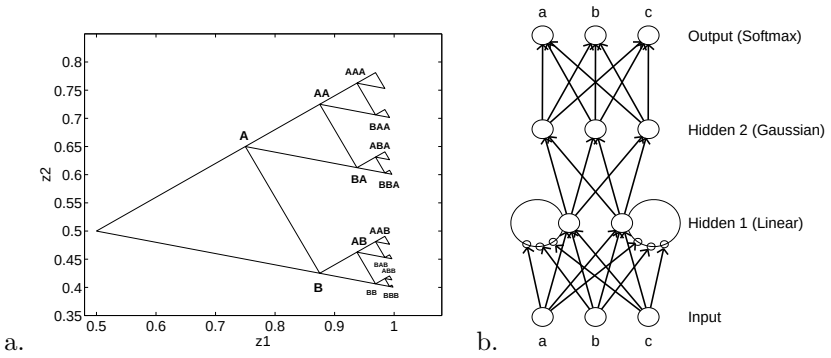


Figure 2: a. Push set for map gleaned by [17]. b. FLNN for Grammar 1.

in the learning model. (ii) Part of the complexity of the solution embodied in equation (1) lies in the use of a rotation when the network crosses the midpoint of a palindrome. (This rotation is accomplished by the submatrix $\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ in the third quadrant of the recurrent weight block in (1)). The rotation has the effect of mapping the whole push-set into a new subspace so the network can easily tell whether it is in push- or pop-mode. This solution is somewhat akin to doing a global symbol substitution across the stack in a PDA. it preserves the stack structure while changing the identity of every element in it. A simpler solution with a PDA is to use a control-state variable to keep track of whether the midpoint has been passed. Indeed, [23] handles such cases with DAs by invoking an additional dimension which encodes the analog of a control state change. Suspecting that learning the rotation or the control state change might be difficult for a network, I have focused here on cases in which a stack alone is sufficient—no separate control variables are needed.

2.3 Fractal Learning Neural Networks (FLNN's): Implementation of Dynamical Automata in RNNs

Figure 2b shows a Fractal Learning Neural Network (FLNN), one way of implementing a dynamical automaton in artificial neurons. The input layer uses a one-hot encoding for words of the vocabulary, as in [4]. The first hidden layer is recurrently connected. The input layer has two kinds of projections to the first hidden layer: first-order connections project directly to the hidden units; second-order connections control the weights on the recurrent hidden connections. The first hidden layer has a linear (identity) activation function. All maps are discrete. The following is a general form of the update rule for the first hidden layer:

$$z_i(t) = \sum_{j \in Inputs} w_{ij} a_j(t) + \sum_{k \in Hidden1} \sum_{j \in Inputs} s_{ikj} z_k(t-1) a_j(t) \tag{3}$$

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. Because the PDDA analysis indicates that only the self-weights need to be manipulated for handling context free grammars 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 "one-hot" codes, 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:

$$z_i(t, j) = w_{ij} + z_i(t-1) s_j \tag{4}$$

Equation (4) should be compared to the equations in the "State Change" column of Table 1, which it implements.

The Hidden1 units have first-order connections to the units in Hidden2. These second hidden layer units have gaussian activation functions:

$$g_i(t) = exp\left[-\frac{|\vec{w}_i - \vec{z}(t)|^2}{b_i^2}\right] \tag{5}$$

Here, g_i is the activation of the i'th second hidden layer unit and b_i^2 is its "bias" (a parameter which controls the radius of the spherical region of Hidden1 space over which the unit is active); $\vec{w}_i$ is the vector of weights feeding from Hidden1 to unit i in Hidden2; $\vec{z}$ is the vector of Hidden1 activations.

The second hidden layer 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):

$$o_i(t) = \frac{exp(net_i(t))}{\sum_{k \in Outputs} exp(net_k(t))} \tag{6}$$

$$net_i(t) = \sum_{j \in Hidden2} w_{ij} g_j(t) \tag{7}$$

The network receives symbols in sequence from the language it is trained on. Each symbol activates a single, unique unit on the input layer. The activations are updated layer-by-layer in the order, Input, Hidden1, Hidden2, Output. In the recurrent layer, Hidden1, all the new states are computed before any of the states are actually changed. The job of the network is to activate on the output layer, after each symbol is presented, the correct probability distribution over next-symbols [4]. The (gaussian) radial basis functions at Hidden2 are used by

Table 3: Grammar 2. (See Table 2 for explanation.)

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)

the network to classify the branches of the fractal on which the network travels in Hidden1. For the languages considered here, each fractal branch in Hidden1 corresponds to a different immediate future and hence a different probability distribution on the output layer. The use of radial basis functions rather than sigmoid functions on Hidden2 helps to drive the formation of the contraction maps which are important for implementing the fractal scaling: if the FLNN persists in using non-contractive scaling, then it will suffer great inaccuracy when multiple embeddings drive it outside the halo of the radial basis units.

3 Simulations

3.1 Training Languages

Matlab script for running simulations like those reported here is available at http://solab.uconn.edu/People/Tabor/papers.html. I used the training languages specified by Grammar 1 (Table 2) and Grammar 2 (Table 3). These languages are exponential state-growth languages. Their state-growth functions are $2^{\left(\frac{L}{3}+1\right)}-1$ and $2^{\left(\frac{L}{2}+1\right)}-1$, respectively [24].

3.2 Training Procedure

Two constraints were put on the FLNN to make learning easier. The three gaussian units in the second hidden layer were assigned fixed variances (here 0.25). These units end up classifying the branches of the fractal. Since the radius of each fractal branch is arbitrary, provided it is positive, the second hidden layer variances can be fixed without loss of generality. All the hidden self-weights were set to 1 initially. The self-weights perform the fractal contraction and expansion, so setting them to the multiplicative identity is the unbiased choice.

The networks were trained by gradient sampling in batch mode. At each learning time step, a sample of small displacements from the current location in weight space is sampled, and the one that reduces the error most is chosen. Gradient sampling is simple to implement and conventional methods of computing the gradient accurately—e.g., Backpropagation Through Time [13, 20, 26, 28, **?**] and Real-time Recurrent Learning [16, 29]—suffer from disappearance of the error signal across long time lags [8]. A disadvantage of the current gradient

sampling method is that it is inefficient and not obviously biologically motivated. Despite these drawbacks, the method is robustly effective in the case of the problems considered here.

The gradient sampling was implemented as follows. A corpus of sentences from the language was processed at the current weight setting and at points on a sphere in weight-space surrounding the current setting. Only a set of basis directions and their negatives was tested. Whichever single weight adjustment produced the greatest reduction in the error relative to the current setting was adopted and the process was repeated. Error on a particular word was measured as Kullback-Leibler Divergence at the output layer ($E = \sum_{i \in Outputs} t_i log(\frac{t_i}{o_i})$, where t_i is the probability of word i in the context and o_i is the activation of output unit i) and was summed over all words in the training corpus. The radius of the sphere was 0.001. The initial values of the Hidden1-to-Hidden2 weights and the Hidden2-to-Output weights in each network were randomly picked from the uniform distribution on (-0.3, 0.3).

The training corpus for Language 1 had one instance of each sentence up to length 9; for language 2, up to length 6—i.e. up to 3 levels of recursion in each case. From the corpus distribution associated with each language, I computed the transition probabilities in an infinite training corpus and used these as the targets on the output layer. The networks were trained until their mean error per word on the test corpus dropped below 0.001 or the gradient became so shallow that it appeared flat in single-precision floating point.

3.3 Training Results

Test Statistics

The test corpora consisted of all sentences of length 12 to 15 words from Language 1 and all sentences of length 8 to 10 words from Language 2 (i.e., novel sentences with ≤ 5 levels of recursion).

FLNN1 and FLNN2 processed their training sets completely accurately in 9 and 11 out of 15 trials, respectively. I will refer to the runs with accurate training set performance in each case as the "successful runs". No unsuccessful runs achieved more than 93% accuracy on the training corpus. To detect and avoid over-learning, the networks were tested on the testing corpus as well as the training corpus throughout training. This precaution turned out not to be necessary: the minimum of testing error occurred at, or very near, the end of training and the difference in performance was slight. All results reported are from the end of training.

Table 4 shows the Root Mean Squared Error [7] and the percentage of correctly predicted transitions for the two networks on its training corpus and test corpus. The very low error rates on the test corpus suggest that each network has successfully generalized to a language with similar structure to that of the target. The analysis of the following section reveals that this performance degrades when deeper levels of embedding are tested, but it also shows that the successful networks have adopted solutions that are structurally analogous to those of

Table 4: Performance on training and test sets for the two FLNN's. RMSE = Root Mean Squared Error. % Cor. = Percent Correct. N = the number of networks that contributed to the computation of Standard Error (SE). Npoints = the number of words tested per network.

Language	Corpus	RMSE	SE	% Cor.	SE	N	Npoints
1	Training	0.013	0.001	100.000	0.000	9	129
1	Test	0.048	0.005	99.424	0.195	9	4755
2	Training	0.008	0.000	100.000	0.000	11	276
2	Test	0.022	0.001	99.900	0.022	11	15232

the corresponding PDDAs. In this sense, they have successfully identified the intended, infinite-state language.

State Space Analysis

In order to process a language correctly, the network must assign states with different futures to different locations in the recurrent hidden unit space (Hidden1). Therefore, it is helpful to divide the Hidden1 states into equivalence classes of states with identical futures under the grammar. Figure 3a provides one view of such a partition. It was generated from the Hidden1 states of a network that learned Language 1 successfully. Each labeled point in the figure marks the mean of the points associated with the stack state that labels it. Figure 3a was derived from the set of all states the network visited while processing the test corpus (the *Visitation Set* of the corpus). The set of vector means over states with common futures derived from this set will be called the *Mean Visitation Set* or *MVS*.). Figure 3a is analogous to Figure 1a. Indeed, it is clear from comparison of the figures that the trained network has discovered an analogous fractal organization to that of the PDDA. The branches of the fractal have been shaded to distinguish the two different immediate futures that are associated with all points but the initial point in Language 1. These two immediate futures are [0.2 a, 0.8 b, 0.0 c] and [0.2 a, 0.0 b, 0.8 c].[1] Every other successful network had an identically structured set of means.

Two important properties distinguish PDDAs from DAs in general: (a) the regions (or *branches*) of the fractal that correspond to different immediate future states are nonoverlapping; (b) symbolic machine cycles correspond to dynamical machine cycles (See online Appendix 1). Peter Tiňo (P.C.) refers to constraint (b) as the Generalized Fixed Point Condition (GFPC). It says, in essence, that in situations where a PDA makes a loop in its state space, the corresponding PDDA will too. To understand the process by which a FLNN develops its solution, it is helpful to examine properties (a) and (b) separately.

[1] In [0.2 a, 0.8 b, 0.0 c], the decimal numbers indicate probabilities and the letters identify next-symbols.

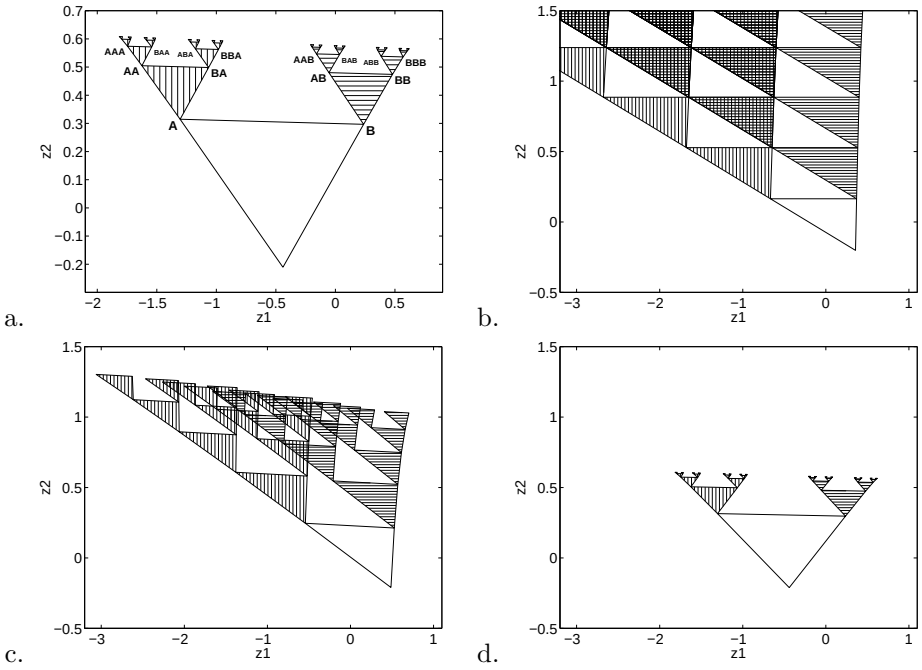


Figure 3: a. State means for a sample FLNN at the end of training. b-d. Evolution of state means during training. The MVS is shown at 10000 steps along the gradient (b), 13500 steps (c) and 25000 steps (d).

a. Overlap of Branches. Figure 3b-d tracks the overlap of fractal branches during the course of training for the same successful network depicted in Figure 3a. In this figure, the stack state labels have been removed to make viewing easier. Figure 3b-d depicts the MVS associated with a test set of all palindromes up to 5 levels of embedding. At the beginning of training, the MVS is a single point because all of the Input-to-Hidden1 weights are initialized to 0. After training has proceeded for a while, the MVS has expanded into an infinite lattice with high overlap between the branches (Figure 3b). This infinite-lattice is the reflection of the unit initial weights in Hidden1, which imply no contraction or expansion. Over the course of training, the branches shrink to a finite size and gradually spread apart (Figure 3c), eventually reaching a nonoverlapping final state (Figure 3d, which shows the same points as Figure 3a). The fact that once the fractal branches come into existence, they monotonically shrink and separate is an indication that the dynamics of the learning process are dominated by an attractor with the topological properties of the PDDA 1 solution.

b. Generalized Fixed Point Condition (GFPC). Figure 4 shows the distribution of points associated with each common future at the end of training. Figure 4a was based on points from the Training Corpus alone (up to 3 levels

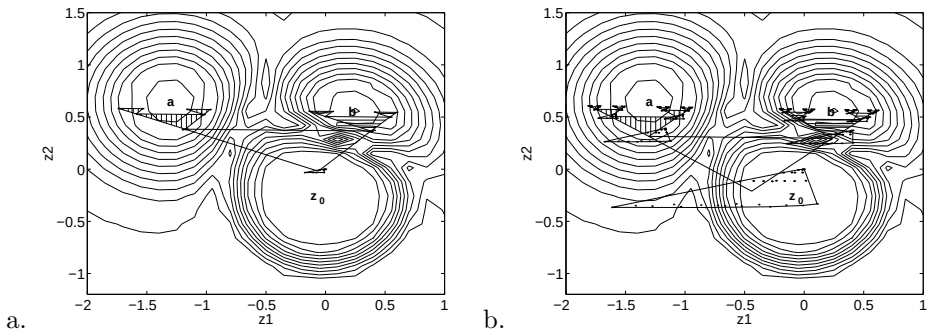


Figure 4: Visited points grouped by source state for (a) 3 levels of recursion (training) and (b) 5 levels (test). The data are from the same run as in Figure 3. Contours show the induced classification surface.

of recursion). In addition to showing the means of points with identical futures at the apices of downward pointing triangles, each set of points with a common future has a polygonal simplex drawn around its periphery. Because the training set was learned with high accuracy, these polygons appear as single points coincident with their means for most levels of recursion. Only the points associated with the 0'th level are visibly separated, and surrounded in the figure by a small triangular simplex. The contours indicate the network's interpretation of the error surface at the end of training. This surface was arrived at by computing the Hidden1-to-Output map for each point, $\vec{z}$ within the plotted region of Hidden1. For each resulting point in Output space, its distance to the three grammar-derived distributions was computed. The height of the induced error surface at $\vec{z}$ was taken to be the minimum of these three distances. Thus the basin structure of Figure 4a indicates how the network classifies points in its Hidden1 space. Indeed, each training point lies well within the basin of the correct probability distribution, in keeping with the fact that the network made no classification errors on the training corpus at the end of training. However, the positive area of the triangles (visible at the 0'th level in the figure) indicates that the network is not perfectly obeying the GFPC: it does not always return to exactly the same point after processing a sequence of symbols that would return the corresponding PDA to the same stack-state.

Figure 4b displays the consequence of this failure in the case of processing more deeply embedded strings. It is identical to Figure 4a except that the Visitation Set on which it is based included the palindromes from each level of recursion up to the maximum level contained in the test corpus (5 levels). Displacements (with respect to the GFPC) at high levels of recursion grow exponentially as the network state scales back up to the 0 level. Thus the set of points that are supposed to coincide with the initial state shows the highest level of distortion. Indeed, some of the initial points and some of the b-predictions land in the wrong basin. Such basin transgressions are the source of the errors

on the test corpora. Although this distortion implies that the network will make many errors on sentences with very deep embeddings, the fact that the network completely separates the branches of the fractal indicates that it has successfully captured the structure of the PDDA solution.

4 Conclusions

Fractal Learning Neural networks appear to be able to reliably discover, via gradient descent exploration, the information structure of a time series generated by several exponential state-growth context free languages. These networks thus show promise of helping to understand, in a principled way, neural learning of complex recursive processes.

Several prior studies have shown that SRNs can approximate natural-language-like recursive patterns [5, 19]. The utility of these findings for researchers of language phenomena will be limited in the long run if we cannot relate them to the structural understanding provided by symbolic analyses. A valuable aspect of the preceding analysis is that it reveals the principle behind the network's solution to the learning problem and clarifies the relationship between network recursion approximation and symbolic perfect recursion.

Several avenues of future work look promising. It may be helpful to pursue a more efficient and accurate method of computing the error gradient—e.g., LSTM [6, 9]. A topic of current work is to improve FLNN generalization accuracy in cases of deep embedding by using the information in the gradient to drive a more perfect approximation of the GFPC. It will also be helpful to consider a wider range of languages than the ones reported on here, including languages that require using a stack with more than two symbols, non-context-free languages, and languages for which a PDA with independent control states is needed, languages with lexical ambiguity (the same word causes different manipulations of the stack, depending on context), and languages with structural ambiguity (the stack state is not uniquely determined by the input).

Much research on neural network learning focuses on the micro-level of the learning mechanism; less has focused on the macro-level of the representations that are to be learned. Indeed, these representations are often inscrutable in nets trained on complex problems. The present results suggest that it is both possible and valuable to study representations in order to guide the search for an effective learning mechanism in complex domains.

Bibliography

[1] Barnsley, Michael, *Fractals Everywhere*, Academic Press Boston (1988).

[2] Bodén, Mikael, and Janet Wiles, "On learning context-free and context-sensitive languages", *IEEE Transactions on Neural Networks* **13**, 2 (2002), 491–493.

[3] Chomsky, Noam, "Three models for the description of language", *IRE Transactions on Information Theory* **2**, 3 (1956), 113–124, A corrected version appears in Luce, Bush, and Galanter, eds., 1965 *Readings in Mathematical Psychology, Vol. 2.*

[4] Elman, Jeffrey L., "Finding structure in time", *Cognitive Science* **14** (1990), 179–211.

[5] Elman, Jeffrey L., "Distributed representations, simple recurrent networks, and grammatical structure", *Machine Learning* **7** (1991), 195–225.

[6] Gers, Felix A., and Jurgen Schmidhuber, "LSTM recurrent networks learn simple context-free and context-sensitive languages", *IEEE Transactions on Neural Networks* **12**, 6 (2001), 1333–1340.

[7] Haykin, Simon S., *Neural networks: a comprehensive foundation*, MacMillan New York (1994).

[8] Hochreiter, Sepp, "The vanishing gradient problem during learning recurrent neural nets and problem solutions", *International Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems* **6**, 2 (1997), 107–116.

[9] Hochreiter, Sepp, and Jürgen Schmidhuber, "Long short-term memory", *Neural Computation* **9**, 8 (1997), 1735–1780.

[10] Holldobler, Steffen, Yvonne Kalinke, and Helko Lehmann, "Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks", *Advances in Artificial Intellegence*. Springer, C 1997 Berlin; New York (1997), pp. 313–324.

[11] Hopcroft, John E., and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Menlo Park, California (1979).

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

[13] Pearlmutter, Barak A., "Gradient calculations for dynamic recurrent neural networks: A survey", *IEEE Transactions on Neural Networks* **6**, 5 (1995), 1212–1228.

[14] Pollack, Jordan, "On connectionist models of natural language processing", Unpublished doctoral dissertation, University of Illinois. (1987).

[15] Pollack, Jordan B., "The induction of dynamical recognizers", *Machine Learning* **7** (1991), 227–252.

[16] Robinson, A. J., and F. Fallside, "The utility driven dynmaic error propagation network", Technical Report CUED/F-INFENG/TR.1, Cambridge Univ. Engineering Department (1987).

[17] Rodriguez, Paul, "Simple recurrent networks learn context-free and context-sensitive languages by counting", *Neural Computation* **13**, 9 (2001).

[18] Rodriguez, Paul, and Janet Wiles, "Recurrent neural networks can learn to implement symbol-sensitive counting", *Advances in Neural Information Processing Systems 10*, (M. Jordan, M. Kearns, and S. Solla eds.). MIT Press Cambridge, MA (1998), pp. 87–93.

[19] Rohde, Douglas, and David Plaut, "Language acquisition in the absence of explicit negative evidence: How important is starting small?", *Journal of Memory and Language* **72** (1999), 67–109.

[20] Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams, "Learning internal represenations by error propagation", *Parallel Distributed Processing, v. 1*, (D. E. Rumelhart, J. L. McClelland, and the PDP Research Group eds.). MIT Press (1986), pp. 318–362.

[21] Siegelmann, Hava, "The simple dynamics of super Turing theories", *Theoretical Computer Science* **168** (1996), 461–472.

[22] Sun, G. Z., H. H. Chen, C. L. Giles, Y. C. Lee, and D. Chen, "Connectionist pushdown automata that learn context-free grammars", *Proceedings of the International Joint Conference on Neural Networks*, (M. Caudill ed.). Lawrence Earlbaum Hillsdale, NJ (1990), pp. 577–580.

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

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

[25] Tiňo, Peter, "Spatial representation of symbolic sequences through iterative function systems", *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans* **29**, 4 (1999), 386–392.

[26] Werbos, Paul J., "Generalization of backpropagation with application to a recurrent gas market model", *Neural Networks* **1**, 4 (1988), 339–356.

[27] Wiles, Janet, and Jeff Elman, "Landscapes in recurrent networks", *Proceedings of the 17th Annual Cognitive Science Conference*, (J. D. Moore and J. F. Lehman eds.). Lawrence Erlbaum Associates (1995).

[28] Williams, R. J., and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories", *Neural Computation* **2** (1990), 490–501.

[29] Williams, Ronald J., and David Zipser, "Gradient-based learning algorithms for recurrent networks", *Backpropagation: Theory, Architectures, and Applications*, (Y. Chauvin and D. E. Rumelhart eds.). Lawrence Erlbaum Associates (1995), pp. 433–486.