

# Learning Exponential State-Growth Languages by Hill Climbing

Whitney Tabor

## Abstract

Training Recurrent Neural Networks on infinite state languages has been successful with languages in which the minimal number of machine states grows linearly with sentence length, but has fared poorly with exponential state-growth languages. A new architecture learns several exponential state-growth languages nearly perfectly by hill climbing.

## Index Terms

context free language, fractal, exponential state-growth, learning, neural network

## I. INTRODUCTION

RECURRENT neural networks (RNN's) of sufficient size can implement Turing machines [1] and thus perform the same computations as symbolic computing mechanisms. But training RNN's to learn infinite-state languages has not been easy. Recently, a number of researchers have succeeded in inducing a stack-like mechanism in an RNN with an undifferentiated set of hidden units [2] [3] [4] [5] [6] [7]. Impressive accuracy has been achieved for languages in which the number of necessary machine states grows linearly with the maximum length of sentence (e.g.,  $a^n b^n$ ,  $a^n b^n c^n$ ). For these languages, it suffices for the hidden units to function as symbol counters [2]. But performance has been much poorer on exponential state-growth languages, which require the machine to keep track of the order, as well as the number of symbols on the stack. Table I shows several languages on which RNN's have been trained, along with their state-growth rates.

The author is with the Department of Psychology, University of Connecticut, Storrs, CT 06269, USA, Phone: (860) 486-4910, email: whitney.tabor@uconn.edu

TABLE I

STATE-GROWTH RATES FOR SEVERAL LANGUAGES. “STATE-GROWTH” IS THE MINIMAL NUMBER OF STATES A SYSTEM MUST DISTINGUISH IN ORDER TO CORRECTLY PROCESS ALL GRAMMATICAL STRINGS OF LENGTH  $\leq L$  (FOR THOSE  $L$  FOR WHICH SUCH STRINGS EXIST).  $wW$  IS THE LANGUAGE IN WHICH AN ARBITRARY SEQUENCE OF WORDS,  $w = w_1w_2\dots w_n$ , MUST BE FOLLOWED BY A EQUAL-LENGTH SEQUENCE,  $W = W_1W_2\dots W_n$ , AND EACH CORRESPONDENCE  $w_i \leftrightarrow W_i$  IS AN INSTANCE OF EITHER  $a \leftrightarrow A$  OR  $b \leftrightarrow B$  (A CROSSED SERIAL DEPENDENCY LANGUAGE).  $wW^R$  IS IDENTICAL TO  $wW$  EXCEPT THAT THE ORDER OF THE  $W_i$  'S IS REVERSED (A PALINDROME LANGUAGE). “X% AT  $L \leq p$ ” MEANS THE NETWORK ACCURATELY SPECIFIED X% OF THE WORD TRANSITIONS IN ALL SENTENCES UP TO LENGTH  $p$ .

Language	State-Growth	Best cited performance	Training Set	Source
$a^n b^n$	$L$	100% at $L \leq 2000$	$L \leq 20$	[5]
$a^n b^n c^n$	$L$	100% at $L \leq 1500$	$L \leq 120$	[5]
$a^n A^m B^m a^n$	$\left(\frac{L}{2} + 1\right)^2 - 1$	100% at $L \leq 92$	$L \leq 44$	[5]
$wW^R$	$3(2^{\frac{L}{2}} - 1)$	$\leq 81\%^1$ at $L = 14$	$L \leq 12$ plus assorted $L \leq 20$	[6]
$wW$	$3(2^{\frac{L}{2}} - 1)$	69% at $L \leq 6$	Grammar Sample <sup>2</sup>	[8]

All the RNN’s mentioned process corpus distributions. 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 *grammatical string* of a corpus-distribution if it has nonzero probability. A machine *correctly processes* a grammatical string from a corpus distribution if, upon being presented with each word of the string in succession, it accurately specifies the probabilities of the words that follow. An output activation vector *accurately specifies* a probability distribution if it is closer to the correct distribution than to any other distribution associated with a grammatical prefix in the corpus-distribution. When only one next-word is possible, this method of assessing correctness is equivalent to the method used in some previous work: count a prediction as correct if the activation of the appropriate output unit is above 0.5 [3] [4] [6] [7]. The current

<sup>1</sup> [6] reports 39 correct out of 114 mixed sequences (i.e., with both a’s & b’s). I concluded that there were a minimum of  $114 - 39 = 73$  errors among at most  $3(2^7 - 1)$  states.

<sup>2</sup> [8] generated strings with a probabilistic queue-grammar in which the probability of pushing a symbol onto the queue was 0.2 at each point. The network was trained on one pass through a 3 million word corpus of such strings.

method has the advantage of also being useful in cases where probabilities are important. Table I cites best published performance levels under this definition. The poor performance of RNN's on exponential state-growth cases is an impediment to using them for natural language processing [8], and for the many applications of symbolic stack machines.

## II. PUSHDOWN DYNAMICAL AUTOMATA AND FRACTAL LEARNING NEURAL NETWORKS

Insight into the challenge of *encoding* arbitrary stack manipulations in RNN's has been provided by [9], [10], [1], [11], and [12] who show that fractal sets provide a natural approach. This paper takes up the learning challenge in the framework of [12], who defines *pushdown dynamical automata* (PDDA's) for processing Context Free Languages (CFL's) in a bounded, real-valued activation space. PDDA's associate words with branches of a multi-dimensional Cantor set. If the branches are non-overlapping, then the machine can process a CFL and there exist such non-overlapping fractals for all CFL's [12].

### A. Network Architecture

*Fractal Learning Neural Networks* (FLNN's) are neural networks that induce PDDA's. Each node on the input layer of a FLNN encodes a word from the vocabulary as in [2]. Each unit in the first hidden layer is self-connected. The input layer projects directly to the first hidden layer units and has second-order connections to their self-weights. These second-order connections are linked so that when a given input unit is on, it specifies the same value of the self-weight on all hidden units. The first hidden layer has a linear activation function (identity) and first-order connections to the second hidden layer. The second hidden layer has a gaussian activation function. It projects to the output units, which, as a group, have the normalized exponential (or "soft-max") activation function, since they model the probabilities of next-words. All maps are discrete.

TABLE II

GRAMMARS 1 AND 2. EACH PRODUCTION IS ASSOCIATED WITH A PROBABILITY. PARENTHESES DENOTE OPTIONAL CONSTITUENTS, WHICH OCCUR WITH PROBABILITY 0.2 IN EVERY CASE.

<b>Grammar 1:</b>	$1.0 S \rightarrow A B C$	$1.0 A \rightarrow a (S)$	$1.0 B \rightarrow b (S)$	$1.0 C \rightarrow c (S)$
<b>Grammar 2:</b>	$0.5 S \rightarrow A B$	$0.5 S \rightarrow X Y$		
	$1.0 A \rightarrow a (S)$	$1.0 B \rightarrow b (S)$	$1.0 X \rightarrow x (S)$	$1.0 Y \rightarrow y (S)$

The network receives words in sequence from the language it is trained on. Each word maximally activates a single, unique unit on the input layer. The job of the network is to activate on the output layer, after each word is presented, the correct probability distribution over next-words. The linear hidden layer makes it possible for the network to make maps that are inverses of one another—a useful ingredient in building a neural implementation of a sequence memory stack [9] [13] [12]. The (gaussian) radial basis functions in the second hidden layer allow the network to map the structurally spherical fractal branches of the first hidden layer to nominal classes which the net can associate (via standard pattern classification) with output probability distributions.

### III. SIMULATIONS

#### A. Training Procedure

An FLNN with 4 input units, 2 linear hidden units, 3 gaussian hidden units, and 4 output units was trained on the exponential state-growth languages specified in Table II. Two constraints made learning easier: (i) the gaussian units were assigned fixed variances ( $\sigma^2 = 0.25$ )—this fixed the radius of the fractal branches without loss of generality; (ii) the self-weights in the first hidden layer were initialized to 1—this choice is unbiased with respect to the fractal expansion and contraction which these weights must perform [12].

The networks were trained by hill climbing in batch mode. A training-corpus of sen-

tences was processed at the current weight setting and at poin

TABLE III

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.

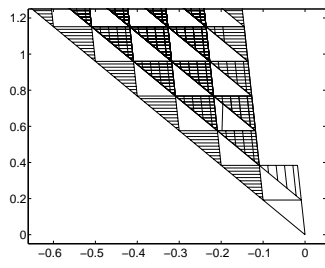
Language	Corpus	RMSE	SE	% Cor.	SE	N	Npoints
1	Training	0.013	0.001	100.000	0.000	9	129
1	Testing	0.048	0.005	99.424	0.195	9	4755
2	Training	0.008	0.000	100.000	0.000	11	276
2	Testing	0.022	0.001	99.900	0.022	11	15232

layer states the FLNN visited while processing sentences involving 8 or fewer stack pushes and no stack pops. The figure shows how stack information is stored by the network while ignoring, for the moment, its retrieval. At the start of training, the input-to-hidden weights were set to 0 so only the hidden state  $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$  was visited. After a small amount of training, the set of visited points has expanded into an infinite lattice (1A). There is much overlap between points associated with the transition  $[0.2 \text{ a}, 0.8 \text{ b}, 0.0 \text{ c}]$  (the “a-branch”) and points associated with  $[0.2 \text{ a}, 0.0 \text{ b}, 0.8 \text{ c}]$  (the “b-branch”).<sup>2</sup> These points must be separated in order for the network to correctly distinguish the states [12]. As training proceeds, the lattice becomes a bounded fractal, and its branches spread apart (1B), eventually separating (1C). What is not shown in Figure 1 is the pop-set (i.e. the set of states that the net inhabits after a series of pushes and exchanges followed by one or more pops). At the end of training, this set is similar to Figure 1C, but it is somewhat displaced from it, because the learning process did not perfectly succeed in making the pops invert the pushes. Such imbalance is the source of the errors that the FLNN’s make on the testing sets.

#### IV. CONCLUSIONS

To my knowledge, these results constitute the first case in which an exponential stack-growth language has been accurately induced by a neural, gradient-following mechanism.

<sup>2</sup> In  $[0.2 \text{ a}, 0.8 \text{ b}, 0.0 \text{ c}]$ , the decimal numbers indicate probabilities and the letters identify next-words.



- [4] M. Bodén and J. Wiles, “Context-free and context sensitive dynamics in recurrent neural networks,” *Connection Science*, vol. 12, no. 3, pp. 197–210, 2000.
- [5] Felix A. Gers and Jurgen Schmidhuber, “Lstm recurrent networks learn simple context-free and context-sensitive languages,” *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1333–1340, 2001.
- [6] Paul Rodriguez, “Simple recurrent networks learn context-free and context-sensitive languages by counting,” *Neural Computation*, vol. 13, no. 9, 2001.
- [7] Mikael Bodén and Janet Wiles, “On learning context-free and context-sensitive languages,” *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 491–493, 2002.
- [8] Whitney Tabor, “The value of symbolic computation,” *Ecological Psychology*, vol. 14, no. 1/2, pp. 21–52, 2002.
- [9] Michael Barnsley, *Fractals Everywhere*, Academic Press, Boston, 1988.
- [10] Cris Moore, “Dynamical recognizers: Real-time language recognition by analog computers,” *Theoretical Computer Science*, vol. 201, pp. 99–136, 1998.
- [11] Peter Tiño, “Spatial representation of symbolic sequences through iterative function systems,” *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 29, no. 4, pp. 386–392, 1999.
- [12] Whitney Tabor, “Fractal encoding of context-free grammars in connectionist networks,” *Expert Systems: The International Journal of Knowledge Engineering and Neural Networks*, vol. 17, no. 1, pp. 41–56, 2000.
- [13] Steffen Holldobler, Yvonne Kalinke, and Helko Lehmann, “Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks,” in *Advances in Artificial Intelligence*, pp. 313–324. Springer, C 1997, Berlin; New York, 1997.