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 faired poorly with exponential state-growth languages. A new architecture learns several exponential state-growth languages nearly perfectly by hill climbing.**

I. Introduction

Recurrent neural networks (RNNs) of sufficient size can implement Turing machines [1] and, thus, perform the same computations as symbolic computing mechanisms. But training RNNs 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]–[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 RNNs have been trained, along with their state-growth rates.

All the RNNs mentioned process corpus distributions.[1],[2] 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 (e.g., [6]). The current 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 RNNs 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.

Manuscript received September 24, 2002.

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

Digital Object Identifier 10.1109/TNN.2003.809421

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

[2]Reference[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.

II. Pushdown Dynamical Automata and Fractal Learning Neural Networks

Insight into the challenge of *encoding* arbitrary stack manipulations in RNNs 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* (PDDAs) for processing context free languages (CFLs) in a bounded real-valued activation space. PDDAs associate words with branches of a multidimensional Cantor set. If the branches are nonoverlapping, then the machine can process a CFL and there exist such nonoverlapping fractals for all CFLs [12].

A. Network Architecture

Fractal Learning Neural Networks (FLNNs) are neural networks that induce PDDAs. Each node on the input layer of an 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.

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 four input units, two linear hidden units, three Gaussian hidden units, and four output units was trained on the exponential state-growth languages specified in Table II. Two constraints made learning easier: 1) the Gaussian units were assigned fixed variances ($\sigma^2 = 0.25$)—this fixed the radius of the fractal branches without loss of generality and 2) the self-weights in the first hidden layer were initialized to one—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 sentences was processed at the current weight setting and at points on a sphere in weight-space surrounding the current setting. Only a set of orthogonal basis vectors and their negatives were tested. Whichever single weight change produced the greatest reduction in the error was adopted and the process repeated. Error was measured as Kullback-Leibler divergence at the output layer. The sphere radius was 0.001. The training corpus for Language 1 consisted

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, $w = w_1 w_2 \ldots w_n$, MUST BE FOLLOWED BY A EQUAL-LENGTH SEQUENCE, $W = W_1 W_2 \ldots W_n$, 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[2]	[8]

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

Grammar 1:	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)
Grammar 2:	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)

TABLE III
PERFORMANCE ON TRAINING AND TEST SETS FOR THE TWO FLNNS. $RMSE$ = 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

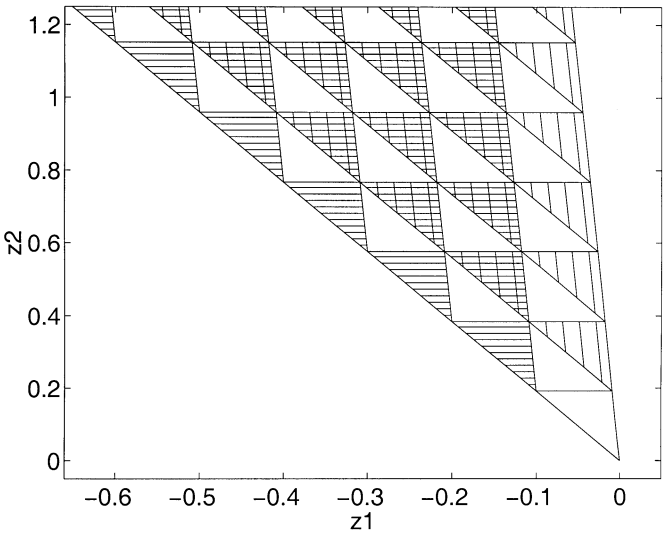
of one instance each of all sentences of length $\leq$ 9; for Language 2, one each of length $\leq$ 6.[3] From the corpus distributions, we derived the word-to-word transition probabilities for the training corpora and used these as the targets on the output layer. The networks were trained until their mean error per word on the training corpus dropped below 0.001 or the gradient became so shallow that it appeared flat in single-precision floating point.

B. Training Results

The testing corpus consisted of all sentences of length 12 to 15 words from Language 1 and all sentences of length eight to ten words from Language 2 (i.e., novel sentences with $\leq$ 5 levels of recursion for each). I tested all sentences within a short depth range in order to find out how close to perfectly implementing the recursive structure the nets were coming on a corpus of manageable size. Nine out of the 15 of the Language 1 networks and 11 out of the 15 Language 2 networks learned successfully in the sense that they achieved 100% accurate specification of the training corpus distribution. Table III shows that the successful networks also made very few incorrect transitions on the test sets.

C. State-Space Analysis

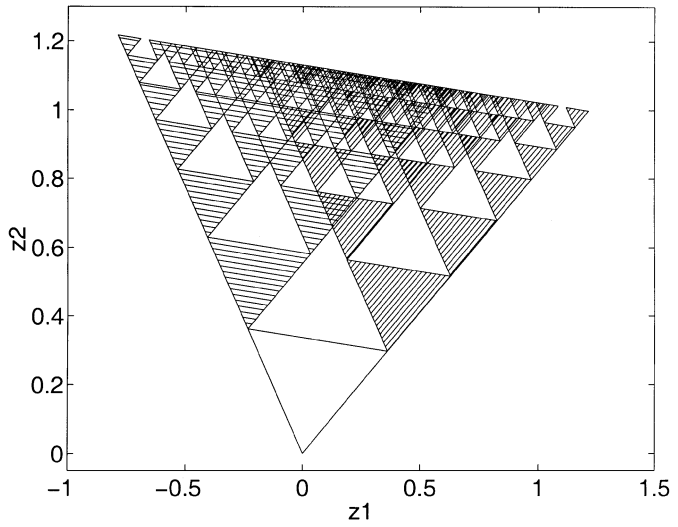
Fig. 1 shows the stages the FLNN goes through. It was generated using the weights of a typical FLNN near the beginning, middle, and end of training, by plotting all first-hidden-layer states the FLNN visited while processing sentences involving eight or fewer stack pushes and no stack pops. The figure 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 $\binom{0}{0}$ was visited. After a small amount of training, the set of visited points has expanded into an infinite lattice [Fig. 1(a)]. There is much overlap between points associated with the transition [0.2 a, 0.8 b,



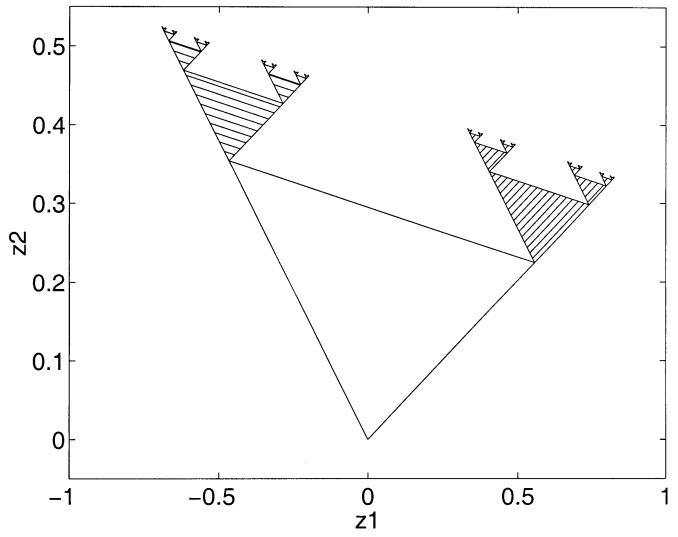
(a)

Fig. 1. Separation of branches in the FLNN during the course of learning. (a) 1000 iterations. $\vec{z} = \binom{z_1}{z_2}$ is the first hidden layer activation vector. It is set to $\binom{0}{0}$ at the start of each sentence. Network states are at the tips of downward-pointing triangles. Approximately-horizontal stripes mark the "a" branch and approximately-vertical stripes mark the "b" branch.

[3]Similar results were obtained when the training sentences formed an unbiased sample from the grammar-defined corpus distribution of each language, but convergence was slower.



(b)



(c)

Fig. 1. (*Continued.*) Separation of branches in the FLNN during the course of learning. (b) 10 000 iterations. (c) 51 174 iterations (the end of training). $\vec{z} = \binom{z_1}{z_2}$ is the first hidden layer activation vector. It is set to $\binom{0}{0}$ at the start of each sentence. Network states are at the tips of downward-pointing triangles. Approximately-horizontal stripes mark the "a" branch and approximately-vertical stripes mark the "b" branch.

0.0 c] (the "a-branch") and points associated with [0.2 a, 0.0 b, 0.8 c] (the "b-branch").[4] 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 [Fig. 1(b)], eventually separating [Fig. 1(c)]. What is not shown in Fig. 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 Fig. 1(c), 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 FLNNs make on the testing sets.

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

IV. Conclusion

To our knowledge, these results constitute the first case in which an exponential stack-growth languages have been accurately induced by a neural gradient-following mechanism. The hidden unit analysis suggests that the network, like a PDDA, is using a fractal to organize its recursive computations. Shortcomings of the method are that the computations are intensive, and nonlocal in time and space. A topic for future work is to test the network on other types of infinite-state languages—e.g., languages involving ambiguity. An advantage of the method is that it couples an effective solution with insight into neural representation. It suggests that techniques in fractal geometry may be useful in developing an analytic understanding of neural learning at the topological, as opposed to the metric, scale.

References

[1] H. T. Siegelmann, *Neural Networks and Analog Computation: Beyond the Turing Limit*. Boston, MA: Birkhäuser, 1999.

[2] J. Wiles and J. Elman, "Landscapes in recurrent networks," in *Proc.e 17th Annu. Cognitive Science Conf.*, J. D. Moore and J. F. Lehman, Eds., 1995.

[3] P. Rodriguez and J. Wiles, "Recurrent neural networks can learn to implement symbol-sensitive counting," in *Advances in Neural Information Processing Systems 10*, M. Jordan, M. Kearns, and S. Solla, Eds. Cambridge, MA: MIT Press, 1998, pp. 87–93.

[4] M. Bodén and J. Wiles, "Context-free and context sensitive dynamics in recurrent neural networks," *Connection Sci.*, vol. 12, no. 3, pp. 197–210, 2000.

[5] F. A. Gers and J. Schmidhuber, "LSTM recurrent networks learn simple context-free and context-sensitive languages," *IEEE Trans. Neural Networks*, vol. 12, pp. 1333–1340, Nov. 2001.

[6] P. Rodriguez, "Simple recurrent networks learn context-free and context-sensitive languages by counting," *Neural Comput.*, vol. 13, no. 9, 2001.

[7] M. Bodén and J. Wiles, "On learning context-free and context-sensitive languages," *IEEE Trans. Neural Networks*, vol. 13, pp. 491–493, Mar. 2002.

[8] W. Tabor, "The value of symbolic computation," *Ecol. Psych.*, vol. 14, no. 1/2, pp. 21–52, 2002.

[9] M. Barnsley, *Fractals Everywhere*. Boston, MA: Academic, 1988.

[10] C. Moore, "Dynamical recognizers: Real-time language recognition by analog computers," *Theoretical Comput. Sci.*, vol. 201, pp. 99–136, 1998.

[11] P. Tiňo, "Spatial representation of symbolic sequences through iterative function systems," *IEEE Trans. Syst., Man, Cybern. A*, vol. 29, pp. 386–392, July 1999.

[12] W. Tabor, "Fractal encoding of context-free grammars in connectionist networks," *Expert Systems: Int. J. Knowledge Eng. Neural Networks*, vol. 17, no. 1, pp. 41–56, 2000.

[13] S. Hölldobler, Y. Kalinke, and H. Lehmann, "Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks," in *Advances in Artificial Intellegence*. Berlin, Germany: Springer, 1997, pp. 313–324.