

Dynamical Parsing to Fractal Representations

Simon D. Levy¹

levys@wlu.edu

Computer Science Department

Washington & Lee University, Lexington, VA 24450, USA

Abstract

A connectionist parsing model is presented in which traditional formal computing mechanisms (Finite-State Automaton; Parse Tree) have direct recurrent neural-network analogues (Sequential Cascaded Net; Fractal RAAM Decoder). The model is demonstrated on a paradigmatic formal context-free language and an arithmetic-expression parsing task. Advantages and current shortcomings of the model are described, and its contribution to the ongoing debate about the role of connectionism in language-processing tasks is discussed.

Introduction

Parsing is a task familiar to both computer scientists and linguists. A popular book on compiler theory and design (Aho, Sethi, and Ullman 1987) describes it thus:

Parsing is the process of determining if a string of tokens can be generated by a grammar. In discussing this problem, it is helpful to think of a parse tree being constructed, even though a compiler may not actually construct such a tree. However, a parser must be capable of constructing the tree, or else translation cannot be guaranteed correct.

For anything but trivial (regular) languages (Hopcroft and Ullman 1979), parsing requires two computational mechanisms: a Finite-State Automaton (FSA) to track input tokens from the string, and a Push-Down Automaton (PDA) or stack, to maintain long-distance dependencies, such as balanced parentheses and subject-verb agreement (Chomsky 1956). If a parse tree is desired, the PDA may also be used to construct it, by pushing and popping grammar-rule productions on and off the stack, a method referred to as “top-down” parsing. For example, Figure 1 shows a parsing automaton (combined FSA and PDA) for the context-free (non-regular) formal language $a^n b^n$ (the language of strings consisting of some number of a s followed by an equal number of b s), as generated a grammar with the productions $S \rightarrow a C$ and $C \rightarrow S b$. The parser yields trees of the form $(a b)$, $((a (a b)) b)$, $((a ((a (a b)) b)) b)$, etc.

Connectionist Parsing

Connectionist (neural network) approaches to parsing have taken a variety of forms. Perhaps the most radical of these, so-called “eliminative” connectionism” (Marcus 1998), seeks to avoid parse trees and other explicit

¹Supported by a grant from the Keck Foundation.

	a	b
S	$S \rightarrow a C$	—
C	$C \rightarrow S b$	$C \rightarrow S b$

Figure 1: Stack automaton for parsing $\{a^n b^n\}$. The automaton is represented as a table that tells the parser what actions to take (push production; pop symbol; shift to next token) based on the current input token and stack top.

structure entirely. Instead, parsing is represented as the ability to predict the next token or token in a sequence or sentence, the idea being that the ability to predict is a central component of parsing. Structure, such as it is, is seen as an emergent property of learning to predict, as evidenced for example by cluster analysis on hidden-layer activations of a Simple Recurrent Network, or SRN (Elman 1990). This approach has been criticized from a number of vantage points, including the observation that prediction is essentially a Markov process², which when done on language corresponds essentially to part-of-speech (POS) tagging, rather than parsing (Steedman 1999).

A second sort of approach essentially builds upon the first, using SRNs, Long Short-Term Memory (Hochreiter and Schmidhuber 1997), and other connectionist architectures to solve practical natural language processing tasks like shallow parsing and named entity extraction (Hammerton 2003), as well as message understanding (Rohde 2002). By casting the parsing problem in practical terms, such approaches have achieved an impressive degree of success in real-world NL tasks, though their relationship to the formal definition of parsing is unclear.

Still other researchers have looked at the problem from a third perspective, namely, how can neural networks be made to perform tasks like language recognition in a way that is mathematically relatable to traditional automata models of such processes. Such work has included the grammatical inference networks of (Giles, Miller, Chen, Chen, Sun, and Lee 1992), the Dynamical Automata of (Tabor 1998), and the the Dynamical Recognizers of (Pollack 1991). All this work shares a focus on traditional concepts like states and transitions, combined with more recent insights from nonlinear dynamical sys-

²C.f. Chomsky’s (1956) observation that Markov processes are inadequate models of natural language (NL), for the reasons cited above.

tems theory. By making use of physically universal concepts like fractal dimension, chaos, and phase transitions, such approaches hold the promise of couching linguistic/cognitive processes within a more general theory of natural phenomena.

The Model

The model presented in the remainder of this paper is based on the third approach described above. Specifically, we will show how the output vectors of a dynamical recognizer can be used to traverse the fractal landscape of an IFS RAAM decoder network (Levy 2002), yielding a model that can parse to dynamically recoverable tree structures while avoiding the use of explicit stacks or push-down automata.

The first component of the model is a Sequential Cascaded Network (SCN), consisting of an input layer, an output layer, a state layer, and recurrent previous-state layer, the latter serving essentially as a second input to represent sequential context. Such a network has been shown to be capable of learning mappings from input sequences to output sequences, including output sequences unspecified for all but a final “accept” or “reject” value – in which case the SCN can be said to induce a finite-state automaton (FSA) for the training set (Pollack 1990). As shown in Figure 2, this network contains both additive and conjunctive (Sigma-Pi) units (Rumelhart, Hinton, and McClelland 1986), resulting in a network whose weights are modulated during input-sequence processing (Pollack 1987).

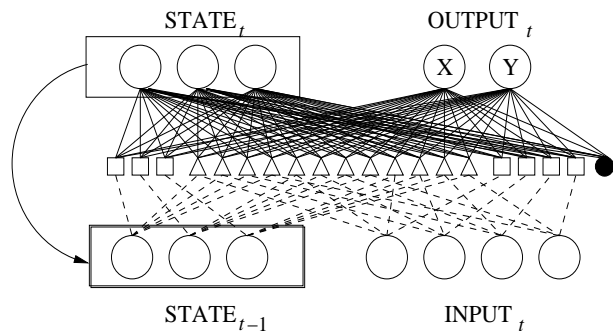


Figure 2: A Sequential Cascaded Network. Dashed lines show direct (square) or multiplicative (triangle) connection to weights. Weights are shown as solid lines. Filled circle represents a bias unit. Output unit labels relate to Figure 3 below. Adapted from (Boden and Wiles 2001)

The second component is an IFS RAAM (IRAAM) Decoder, essentially the hidden and output layers of Pollack’s (1990) Recursive Auto-Associative Memory (RAAM) network. As demonstrated in (Levy 2002), separating the decoder component of the RAAM network and treating it as an iterative function system (IFS) effectively solves the capacity limitations of original RAAM, resulting in a network that can generate an unbounded number of strings of a (non-regular) context-free language (CFL; Melnik, Levy, and Pollack 2000). The key insight of IRAAM is that every point in the vector space of the hidden layer is either part of the fractal attractor of

the IFS, or has a tree-shaped transient (orbit) that “ends up” on the attractor in a finite number of steps. In other words, IRAAM provides a direct mapping from (fixed-width) vectors to arbitrarily-sized trees. Figure 3 shows an IRAAM decoder with two hidden units, and Figure 4 shows an X/Y map of several thousand trees generated by one such decoder that “learned” an interesting attractor under human supervision. IRAAM network weights can be derived adaptively in this manner, or can be “hand-coded” in the manner of (Tabor 1998), as we describe in the next section.

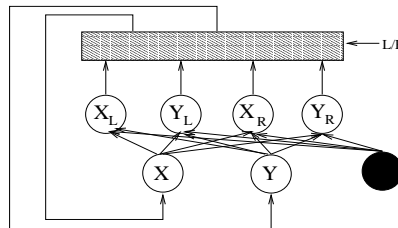


Figure 3: IFS RAAM decoder network with two hidden units. Bar at top of figure is a “gate” that feeds the left (L) or right (R) output of the decoder back onto the hidden layer. Filled unit represents bias.

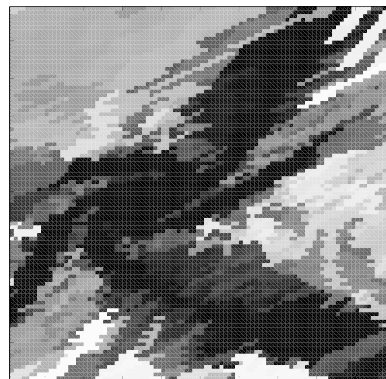


Figure 4: Example map of unique trees in 2D IFS RAAM, sampled at 100x100 pixels from the unit square. Each non-terminal tree is represented by a different grayscale value. Attractor (terminal set) is black spiral.

Experiment 1: Learning to Parse a Simple CFL

To test the feasibility of the SCN/IRAAM parsing model, we first generated IRAAM weights sufficient to decode several parse trees for the $a^n b^n$ grammar described in the Introduction.³ As discussed in (Melnik, Levy, and Pollack 2000), network weights for this grammar can either

³Because of the need to represent intermediate parse trees, this IRAAM is a technically decoder for parse trees over the language $\{a^n b^n \cup a^n b^{n+1} \cup a^{n+1} b^n\}$, though of course this language has the same computational complexity as $\{a^n b^n\}$.

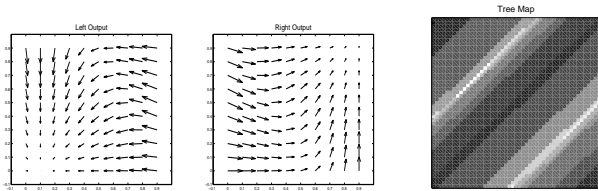


Figure 5: Dynamics and tree map for the $a^n b^n$ decoder. On the left output, points go to the left side and then down to the corner, where there is an attractor point, corresponding to the terminal tree *a*. On the right output, points go rightward and then up to the other attractor point, corresponding to terminal *b*. This symmetry produces the linear pattern in the tree map.

be learned through a method like hill-climbing, or can be hand-coded using a mathematical formula based on a dynamical-systems analysis of the IFS. For this experiment, we chose the latter approach. The resulting network has the dynamics and tree map shown in Figure 5. Unlike the decoder whose tree-map is shown in Figure 4, this network is operating in “saturation”, with extreme weights that yield a simple linear tree map. Hence it provides an ideal, if overly simplistic, target for our initial parsing experiments.

Having obtained the mapping from two-dimensional vectors to parse trees, we exploited the redundancy (linearity) in the tree map, scanning horizontally along the tree map to obtain 10 parse trees. This allowed us to constrain the SCN to have only one output unit, corresponding to the dimension that was varied to obtain the parse trees. For the SCN’s input (tokens) we used an orthogonal code ($[0 \ 1] = a$; $[1 \ 0] = b$), so that the network had two input units. We used Pollack’s “backspace” variant on back-propagation-through-time (Pollack 1990) to train the SCN to map from the input sequences to the corresponding vectors in the tree map; *i.e.*, to map from input strings to parse-tree vectors in the language $\{a^n b^n, n \leq 8\}$. We used five state units, learning rate $\eta = 0.1$, and momentum $\mu = 0.9$. After 10,000 epochs of back-propagation, the SCN was able to produce the correct parse-tree vectors.

Experiment 2: Learning to parse arithmetic expressions

A more practical parsing task, often given as an exercise in compiler courses, involves translating arithmetic expressions into pre- or postfix form; for example, translating the expression $a * (b + c)$ into the tree $(* \ a \ (+ \ b \ c))$.⁴

To model this task in our SCN / IRAAM framework, we first hand-coded the weights for an IRAAM decoder with left, middle, and right outputs (two hidden units, six output units), corresponding to the operator and two operands of a parse tree in prefix form, as in the LISP programming language. For the middle and right out-

⁴Note that the parentheses in the expression are actual input tokens, whereas in the tree they are only a notation to indicate constituency.

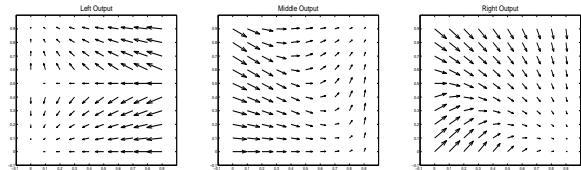


Figure 6: Dynamics for the arithmetic-tree decoder. The left output is a saddle: points above the line $y = 0.5$ jump to $(0,1)$; *i.e.*, terminal $+$, and points below the line jump to $(0,0)$; *i.e.*, terminal $*$. The middle and right outputs have complementary dynamics similar to those of the $a^n b^n$ decoder.

puts we used a variant of the $a^n b^n$ decoder weights, with attractor points in the lower- and upper-right corners of the unit square. For the left output we used weights that took every point directly to the lower- or upper-left corner of the square, guaranteeing that the left branch of any (sub)tree would always terminate immediately; *i.e.*, we wanted to avoid ill-formed parse trees like $((+ \ a \ b) \ a \ b)$. The dynamics of this network are shown in figure 6.

We trained a four-input SCN to parse 12 strings over the alphabet $\{a, b, +, *\}$ to the appropriate parse-tree vector. These strings included instance like $(a + b) * b$, which contains parentheses to override the order of operations, and $a + a * b$, which does not. Using eight state units, the network was able to learn the task under the same conditions as in the previous experiment (10K epochs, $\eta = 0.1$, $\mu = 0.9$).

Conclusions and Future Work

We have presented a rudimentary connectionist model of parsing using a combination of a Sequential Cascaded Network (SCN) and an IFS RAAM (IRAAM) network, which work together to yield a parse tree for a given input string over a small alphabet. By using two different networks, we were able to separate the alphabet of the input stream from the terminal set of the parse trees, a common situation in parsing programming languages and in parsing directly from natural-language input to abstract semantic representations.⁵

The novel contribution of this work is the combination of two distinct dynamical systems (recurrent neural networks), an SCN and an IRAAM, to accomplish the single task of parsing. This division of labor serves the same purpose as it does in the the corresponding discrete (finite-state and push-down) automata; however, we are at pains to emphasize that our model is by no means a mere implementation of such mechanisms (Fodor and Pylyshyn 1988). Not only does our model lack the discrete states of classical automata⁶; it also, more crucially, lacks anything directly corresponding to the *rules* (grammar productions and finite-state transitions) of the classical architecture. In our opinion, it is precisely this ability

⁵*e.g.*, *The boy loves the girl* \rightarrow loves(boy123, girl456)

⁶other than those created by the temporally discrete nature of the input tokens and by implementation on digital computers

to learn and exhibit rule-like behavior in the absence of rules that makes connectionist models worth studying.

Obviously, the ability to parse a few simple expressions is a long way off from the demands placed on programming-language compilers, let alone full-blown NLP. Nevertheless, fields like theoretical linguistics have made tremendous progress by focusing on a variety of specific phenomena using very constrained models, without feeling the need to provide, *e.g.*, a grammar or parser for the entire English language.

We see several possible directions for the work described here. One feature of the model is that it always yields a parse, regardless of how “ungrammatical” the input may be. This is both an advantage and a disadvantage. Real (human) parsers function quite well in the presence of ill-formed input, and grammaticality judgments may be seen as more of an epiphenomenon of parsing than a central feature. On the other hand, it is often possible to scramble an utterance into incomprehensible “word salad” that listeners will reject out of hand. Therefore, we would like our model to provide some mechanism for rejecting completely ill-formed inputs, or even better, to support graded degrees of acceptability. The state space of the SCN component is the natural place to look for such phenomena, though it is not immediately clear how to train the network to avoid certain parts of the space, without providing the negative exemplars that are considered to be crucially absent in real language learning (Chomsky 1965).

Another notable aspect of natural language parsing is the interaction of top-down and bottom-up influences, as evidenced for example by the processing of so-called “garden-path” sentences (Crain and Steedman 1985). In other work (Levy, Melnik, and Pollack 2000) we have shown how the IFS RAAM decoder can be inverted to produce parse trees from the bottom up, by analogy with classical unification algorithms. It is intriguing to speculate how the top-down parsing described in the present paper might be combined with such an approach, in an attempt to model this sort of linguistic behavior.

References

- Aho, A., R. Sethi, and J. Ullman (1987). *Compilers: Principles, Techniques and Tools*. Addison-Wesley.
- Boden, M. and J. Wiles (2001). Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science* 12.
- Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on information theory* 2, 113–124.
- Chomsky, N. (1965). *Aspects of the Theory of Syntax*. Cambridge, Mass.: MIT Press.
- Crain, S. and M. Steedman (1985). On not being led up the garden path: The use of context by the psychological syntax processor. In D. R. Dowty, L. Karttunen, and A. M. Zwicky (Eds.), *Natural language parsing: Psychological, computational, and theoretical perspectives*. Cambridge: Cambridge University Press.
- Elman, J. (1990). Finding structure in time. *Cognitive Science* 14, 179–211.
- Fodor, J. and Z. Pylyshyn (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition* 28, 3–71.
- Giles, C., C. Miller, D. Chen, H. Chen, G. Sun, and Y. Lee (1992). Learning and extracting finite-state automata with second-order recurrent neural networks. *Neural Computation* 4(3), 393–405.
- Hammerton, J. (2003). Shallow parsing with long short-term memory. In *Proceedings of the Seventh Joint Conference on Information Sciences*, Duke University, Durham, NC.
- Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural Computation* 9(8).
- Hopcroft, J. and J. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley.
- Levy, S. (2002). *Infinite RAAM: Initial Explorations into a Fractal Basis for Cognition*. Ph. D. thesis, Brandeis University, Waltham, Mass.
- Levy, S., O. Melnik, and J. Pollack (2000). Infinite raam: A principled connectionist basis for grammatical competence. In *Proceedings of the 22nd Annual Meeting of the Cognitive Science Society*.
- Marcus, G. (1998). Rethinking eliminative connectionism. *Cognitive Psychology* 37, 243–282.
- Melnik, O., S. Levy, and J. Pollack (2000). RAAM for an infinite context-free language. In *Proceedings of the International Joint Conference on Neural Networks*, Como, Italy. IEEE Press.
- Pollack, J. (1987). Cascaded back-propagation on dynamic connectionist networks. *Proceedings of the Ninth Annual Cognitive Science Society Meeting*, Seattle, WA, pp. 391–404.
- Pollack, J. (1990). Recursive distributed representations. *Artificial Intelligence* 36, 77–105.
- Pollack, J. (1991). The induction of dynamical recognizers. *Machine Learning* 7, 227–252.
- Rohde, D. (2002). *A Connectionist Model of Sentence Comprehension and Production*. Ph. D. thesis, Carnegie Mellon University.
- Rumelhart, D., G. Hinton, and J. McClelland (1986). A general framework for parallel distributed processing. In D. Rumelhart and J. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1. MIT.
- Steedman, M. (1999). Connectionist sentence processing in perspective. *Cognitive Science* 23(4), 615–634.
- Tabor, W. (1998). Dynamical automata. Technical Report TR98-1694, Computer Science Department, Cornell University.