Infinite RAAM: A Principled Connectionist Substrate for Cognitive Modeling

Simon Levy and Jordan Pollack
levy, pollack@cs.brandeis.edu
Dynamical and Evolutionary Machine Organization
Volen Center for Complex Systems,
Brandeis University, Waltham, MA 02454, USA

March 1, 2001

Abstract

Unification-based approaches have come to play an important role in both theoretical and applied modeling of cognitive processes, most notably natural language. Attempts to model such processes using neural networks have met with some success, but have faced serious hurdles caused by the limitations of standard connectionist coding schemes. As a contribution to this effort, this paper presents recent work in Infinite RAAM (IRAAM), a new connectionist unification model. Based on a fusion of recurrent neural networks with fractal geometry, IRAAM allows us to understand the behavior of these networks as dynamical systems. Using a logical programming language as our modeling domain, we show how this dynamical-systems approach solves many of the problems faced by earlier connectionist models, supporting unification over arbitrarily large sets of recursive expressions. We conclude that IRAAM can provide a principled connectionist substrate for unification in a variety of cognitive modeling domains.

Language and Connectionism: Three Approaches

Language, to a cognitive scientist, can be held to include natural language and the "language of thought" (Fodor 1975), as well as symbolic programming languages developed to simulate these, like LISP and Prolog. Attempts to build connectionist models of such systems have generally followed one of three approaches.

The first of these, exemplified by (Rumelhart and McClelland 1986), dispenses entirely with traditional representations (data structures) and rules (algorithms on those structures), in favor of letting the network "learn" the patterns in the data being modeled, via the well-known back-propagation algorithm (Rumelhart, Hinton, and Williams 1986) or a similar training method. This approach became the subject harsh criticism from members of the traditional "symbols-and-rules" school of cognitive science, based on the disparity between the strength of the claims made and the actual results reported (Pinker and Prince 1988), as well as the apparent inability of such systems to handle the systematic, compositional aspects of linguistic meaning (Fodor and Pylyshyn 1988).

The second sort of connectionist approach goes beyond the rules-and-representations view and directly to the heart of what computing actually means, by showing how a recurrent neural network can perform all the operations of a Turing machine, or more (Siegelmann 1995). Though such proofs may hold a good deal of theoretical interest, they do not address the degree to which a particular computational paradigm (connectionism) is suited to a particular real-world task (language). They are therefore not of much use in arguing for or against the merits of connectionism as a model of any particular domain of interest (Melnik 2000), any more than knowing about Turing equivalence will help you in choosing between a Macintosh and a Pentium-based PC.

The third approach, which some of its proponents have described as "Representations without Rules" (Horgan and Tienson 1989), is the one that we wish to take here. This approach acknowledges the need for systematic, compositional structure, but rejects traditional, exceptionless linguistic rules in favor of the flexible computation afforded by connectionist representations. Proponents of such a view are of course responsible for showing how these representations can support the kinds of processes traditionally viewed as rules. In the remainder of this paper we show how the behavior of neural network called an Infinite RAAM corresponds directly to one such process, unification, thereby supporting a systematic, compositional model of linguistic structure.

Unification

Unification, an algorithm popularized by Robinson (1965) as a basis for automated theorem-proving, has come to play a central role in both computer science and cognitive science. In computer science, unification is at the core of logical programming languages like Prolog (Clocksin and Mellish 1994); in cognitive science, it is the foundation of a number of category-based approaches to the analysis of natural language (Shieber 1986). The basic unification algorithm can be found in many introductory AI textbooks (e.g., Rich and Knight 1991 p. 152), and can be summarized recursively as follows: (1) A variable can be unified with a literal. (2) Two literals can be unified if their initial predicate symbols are the same and their arguments can be unified.

If, for example, we have a Prolog database containing the assertion `male(albert)`,[1] meaning "Albert is male", and we perform the query `male(Who)`, asking "Who is male?" the unification algorithm will first attempt to unify `male(albert)` with `male(Who)`, and will succeed in matching on the predicate symbol `male`, by rule (2). The algorithm will then recur, attempting to unify the variable `Who` with the atomic literal `albert`, and will succeed by rule (1) and terminate, with the result that `Who` will be bound to `albert`, answering the query.

[1]Prolog examples are taken from the tutorial introduction in (Clocksin and Mellish 1994).

Of course, real programming-language and natural-language applications require unification algorithms more complicated than the one illustrated in this simple example, but the example suffices for our goals here.

RAAM

Before describing how the Infinite RAAM model is suited to performing unification, some historical background on this model is necessary.

Recursive Auto-Associative Memory or RAAM (Pollack 1990) is a method for storing tree structures in fixed-width vectors by repeated compression. Its architecture consists of two separate networks: an encoder network, which can construct a fixed-dimensional code by compressively combining the nodes of a symbolic tree from the bottom up, and a decoder network which decompresses this code into its two or more components. The decoder is applied recursively until it terminates in symbols, reconstructing the tree. These two networks are simultaneously trained as an autoassociator (Ackley, Hinton, and Sejnowski 1985) with time-varying inputs. If the training is successful, the result of bottom up encoding will coincide with top-down decoding. Figure 1 shows an example of a RAAM for storing binary trees using two bits of representation for each input and output.[2]

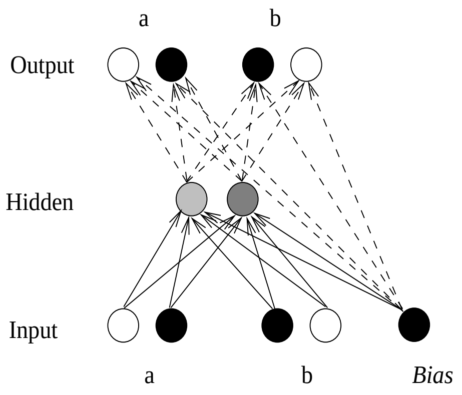

Figure 1: RAAM encoding and decoding the tree `(a b)`, using two bits per symbol: `a = 01`, `b = 10`. Solid lines depict encoder weights, dashed lines decoder weights. Note the real-valued representation of the tree `(a b)` on the hidden layer, which would be fed back into the encoder to build a representation of the trees `(a(a b))`, `(b(a b))`, `((a b)a)`, etc.

Following the publication of (Pollack 1990), RAAM gained widespread popularity as a model of linguistic structure. Some researchers (Blank, Meeden, and Marshall 1991) found it an attractive way of "closing the gap" between the symbolic and sub-symbolic paradigms in cognitive science. Others (Van Gelder 1990) saw in RAAM a direct and simple refutation of the traditional cognitive scientists' backlash against connectionism, or went as far as to show how traditional syntactic operations like transformations could be performed directly on RAAM representations (Chalmers 1990).

RAAM as an Iterated Function System

Consider the RAAM decoder shown in Figure 2. It consists of four neurons that each receive the same (X, Y) input. The output portion of the network is divided into a right and a left pair of neurons. In the operation of the decoder the output from each pair of neurons is recursively reapplied to the network. Using the RAAM interpretation, each such recursion implies a branching of a node of the binary tree represented by the decoder and initial starting point. However, this same network recurrence can also be evaluated in the context of dynamical systems. This network is a form of *iterated function system* (IFS) consisting of two *transforms*, which are iteratively applied to points in a two-dimensional space.

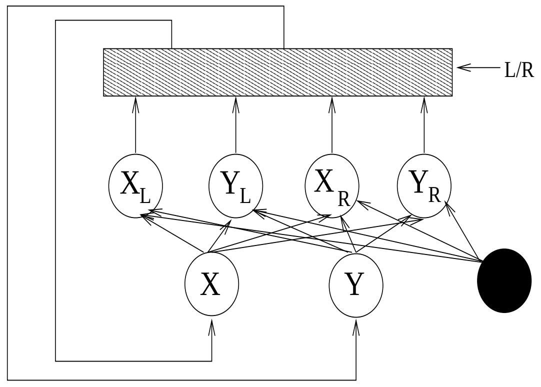

Figure 2: Detail of the decoder from the RAAM of Figure 1. Bar at top of figure is a "gate" that feeds the left or right output of the decoder back onto the hidden layer.

In a typical IFS (Barnsley 1993), the transforms are linear equations of the form $T_i(x) = A_i x + b_i$, where x and b are vectors and A is a matrix. The *Iterated* part of the term IFS comes from the fact that, starting with some initial x, each of the transforms is applied iteratively to its own output, or the output of one of the other transforms. The choice of which transform to apply is made either deterministically or by non-deterministic probabilities associated with each transform. If the transforms T_i are *contractive*, meaning that they always decrease the distance between any two input vectors x and y, then the limit of this process as the number of iterations N approaches infinity yields an *attractor* (stable fixed-point set) for the IFS. Most IFS research has focussed on systems whose attractor is a *fractal*, meaning that it exhibits self-similarity at all scales. [3]

The transforms of the RAAM decoder have the form $T_i(x) = f(A_i x + b_i)$, where f is the familiar logistic-sigmoid "squashing" function $f(x) = 1/(1 + e^{-x})$. Typical of connectionist models, the matrix A ranges over the entire set of

[2]Restricting the network to only two bits per symbol allows straightforward visualization of its hidden-layer dynamics as an X/Y plot. RAAMs for real-world tasks would use many more bits per symbol.

[3]A famous example of a fractal attractor is the beautiful Mandelbrot set, in which tiny copies of the entire set seem to appear as if by magic when you zoom in on certain regions.

real numbers, so it is not necessarily contractive. Nevertheless, the squashing function provides a "pseudo-contractive" property that yields an attractor for the decoder. In the context of RAAMs, however, the main interesting property of (pseudo-)contractive IFSes lies in the trajectories of points in the space. For such IFSes the space is divided into two sets of points. The first set consists of points located on the underlying fractal attractor of the IFS. The second set is the complement of the first, points that are not on the attractor. The trajectories of points in this second set are characterized by a gravitation towards the attractor, as follows: Each iteration produces a set of left and right copies of the points from the previous iteration. Finite, multiple iterations of the transforms have the effect of bringing the set of copies arbitrarily close to the attractor.

Taking the terminal test of the decoder network to be "on the attractor" solves a number of technical problems that limited the scalability of the RAAM model, and allows the model to represent extremely large sets of trees in small fixed-dimensional neural codes. The attractor, being a fractal, can be generated at arbitrary pixel resolution. In this interpretation, each possible tree, instead of being described by a single point, is now an *equivalence class* of initial points sharing the same tree-shaped trajectories to the fractal attractor.

Using the attractor as a terminal test also allows a natural formulation of assigning labels to terminals. Barnsley (1993) noted that each point on the attractor is associated with an address which is simply the sequence of indices of the transforms used to arrive on that point from other points on the attractor. The address is essentially an infinite sequence of digits. Therefore to achieve a labeling for a specific alphabet we need only consider a sufficient number of significant digits from this address.

These ideas are encapsulated in Figure 3, which shows a "Galaxy" attractor obtained by iterative Blind Watchmaker selection (Dawkins 1986) to a visually appealing shape,[4] along with sample derivations of the trees `(a b)` and `(a (a b))`.

Infinite RAAM

Using the "on-the-attractor" terminal test, we were able to use hill-climbing to train a RAAM decoder to generate all and only the strings in the set $a^n b^n \cup a^n b^{n+1}, n \leq 5$. As the simplest example of a non-regular, context-free formal language, $a^n b^n$ has been used as a target set by a number of recurrent-network research projects (Rodriguez, Wiles, and Elman 1999; Williams and Zipser 1989), so it serves as a benchmark for the formal power of a model such as RAAM.

Analysis of the decoder weights of our $a^n b^n$ RAAM revealed a pattern that we were able to generalize into a formal constructive proof for deriving a set of weights to generate this language for arbitrarily large values of n, as a function of the pixel resolution ϵ (Melnik, Levy, and Pollack 2000).

With this proof in hand, we felt justified in using the term *Infinite RAAM* (IRAAM) to refer to our decoder networks. Against a traditional approach in which grammars are the only sufficient competence models and neural networks are merely implementations (Fodor and Pylyshyn 1988) or performance models, the formally proven existence of a set of "pure" $a^n b^n$ weights provided evidence that a neural network can serve as both a competence model and a performance model, under a dynamical-systems interpretation of the network's behavior.

[4] A gallery of several such attractor images can be viewed at **http://demo.cs.brandeis.edu/pr/mindeye/bwifs.html**

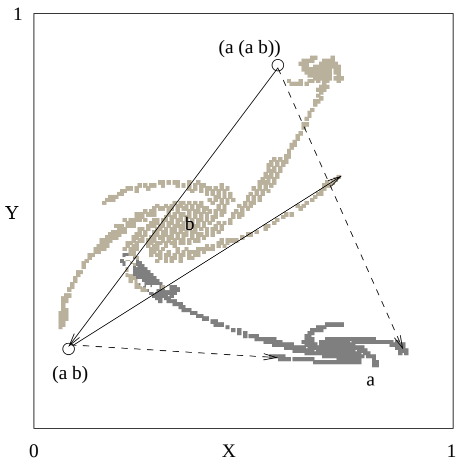

Figure 3: The Galaxy attractor, showing derivation of the tree `(a (a b))` and its daughter tree `(a b)`. Attractor points with addresss `a`, reachable from the attractor on the left transform, are colored dark gray; points with address `b`, reachable on the right transform, are light gray. The left transients to the attractor are shown as dashed lines, and the right transients as solid lines.

Unification-based IRAAM

Nevertheless, a fundamental problem exists in the general case when investigating the capacity of a given IRAAM decoder via discrete sampling of the space of tree equivalence classes. Transients to the attractor can potentially meander around the entire unit space before coming to rest on the attractor, so the potential depth of the trees encoded using even a low-resolution sampling is quite large. Because the number of possible trees grows factorially with the depth of the trees, the discrete sampling method is therefore doomed to find only an infinitesimal portion of the trees that a given IRAAM could be encoding. Solving this problem requires knowing precisely how many trees to search for, and where to find them.

To limit the number of trees, it is sufficient to limit the number of IFS iterations. Like sampling, limiting the iterations produces only an approximation to the actual, infinite attractor. For zero iterations, the entire space is the attractor approximate, and the only tree encoded is a terminal, which we may refer to generically as `X`. For one iteration, each point not on the attractor goes to the attractor on one iteration, and the only tree encoded is `(X X)`. For two iterations, the trees encoded are `(X (X X))`, `((X X) X)`, and `((X X) (X X))`, and so on for more iterations. This solves the first part of the problem.

Solving the second part of the problem – locating the trees in space – requires switching from a "top-down" approach to a "bottom up" approach. We no longer start at a point off the attractor and decode the tree as this point's path to the attractor. Instead, we start at a point (or set of points) on the attractor, and ask what other point(s) that point can be unified with, using the *en*coder: hence the term *unification-based* IRAAM.[5]

To perform this unification, we first compute the attractor, then take its image under the left and right *inverses* of the transforms. Unifications (trees) are located precisely within the intersections of these inverses. Under this interpretation, asking whether two constituents can be unified means asking whether their inverses have a non-empty intersection.

For example, to determine the locations of the binary trees of depth two or less, we iterate the IFS twice, producing four attractor pieces, each of which is an image of the unit square under the composition of two transforms (left/left, left/right, right/left, and right/right). The union of these is the attractor approximate A_2, which encodes the abstract terminal tree `X`. This process is shown in Figure 4.

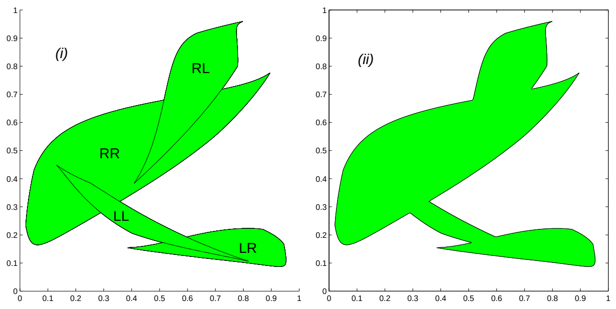

Figure 4: The Galaxy attractor approximated at two IFS iterations: individual overlapping images (*i*) and union of these images (*ii*). The union represents the outer boundary of the sampled attractor shown in Figure 3.

Taking the left and right inverse transforms of the attractor in Figure 4 gives us the regions shown in Figure 5. Intersecting these regions gives us the region encoding the trees `X` and `(X X)`. As depicted in Figure 6, "subtracting out" the attractor (tree `X`) gives us the region encoding only the tree `(X X)`.

At this point we have everything we need to encode the remaining trees of depth two. To encode the tree `(X (X X))`, we take the left inverse of the attractor and intersect this inverse with the right inverse of the region encoding the tree `(X X)`. This right inverse is the entire unit square [6], so this intersection is effectively a no-op, giving us the same left inverse that we started with. Subtracting out the trees `(X X)` and `X`, which are contained in this inverse, gives us the region encoding only the tree `(X (X X))`. Swapping "left" for "right", the same operations can be done to obtain the tree `((X X) X)`; neither of these is shown, to save space. Finally, the tree `((X X) (X X))` is encoded by the region not encoding any of the other trees.

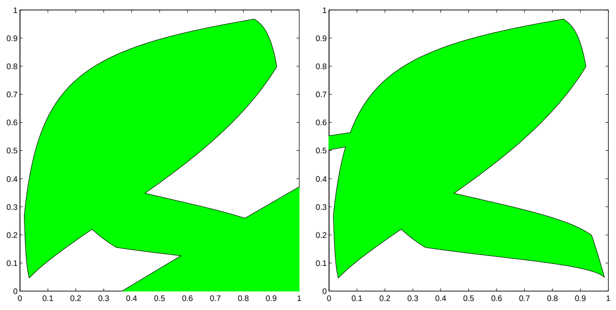

Figure 5: Left and right inverses of the attractor of Figure 4. These inverses encode the trees `(X (X X))` and `((X X) X)`; see text.

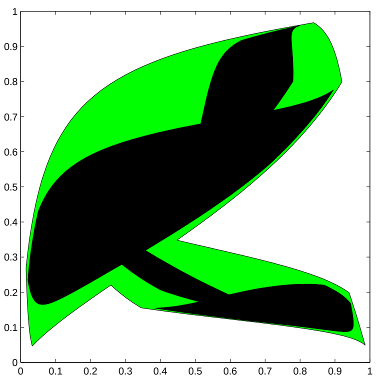

Figure 6: Intersection of inverse regions from Figure 5, which encodes the tree `(X X)`. Black region is the attractor, which is "subtracted out" from the intersection.

Labeling the Terminals

The discussion of the hill-climbing $a^n b^n$ decoder described a scheme for labeling the points of the attractor terminal set by means of their fractal addresses. The method involved approximating the attractor at some pixel resolution, then labeling each attractor point by the transform(s) on which that point was reachable from any points on the attractor. This scheme cannot be implemented in a model in which the attractor is approximated by iteration, because the only points reachable from the current attractor approximate A_N lie on the approximates $A_{N+k}, k > 1$. Since these points themselves are not on the part of A_N reachable from outside A_N , this scheme cannot be used to label the terminals of trees, which by definition are transients the to attractor from points outside it.

For the current stage of the project, we are working around this problem by simply hand-labeling regions in the attractor, as illustrated in the simple database example below.

[5]We find a compelling parallel in the historical switch from "top-down" Chomskyan rules (Chomsky 1957) to the "bottom-up" combinatorial categories of unification-based grammars (Shieber 1986)

[6]because we have performed the operation $R^{-1}(R^{-1}(R(R(\square))))$

Bringing it All Together

By this point we hope to have persuaded the reader that IRAAM provides a plausible connectionist substrate for unification-based models. To make this point more concrete, we can consider how one would implement a simple logical database language, like first-order predicate calculus (FOPC), or Prolog, using an IRAAM.

Labeled attractor regions correspond to *atoms*: `albert`, `victoria`, `female`. *Facts* about atoms, like `female(victoria))` and `parents(edward, victoria, albert)`, are built recursively as intersections of the inverses of the labeled attractor regions, and intersections of the inverses of those intersections. Whether or not two constituents (atom, propositions) can be unified depends on the size and shape of their encoding regions, and on the connection weights of the IRAAM network.

Rules relating facts to each other and generalizing them using variables, correspond to intersections or unions of the recursively constructed facts. For example, the rule `woman(X) :- human(X), female(X), adult(X)` would be implemented by taking the intersection of the inverses of the attractor regions for the atoms `human`, `female`, and `adult`; this intersection would be the "definition" of the term `woman` in the model.

We can illustrate this process using our toy database, presented below in its entirety:

```
male(albert).
male(edward).
female(alice).
female(victoria).
parents(edward, victoria, albert).
parents(alice, victoria, albert).
```

Encoding these propositions in a format that a binary-branching IRAAM could represent requires some slight modifications: all propositions are first put in prefix form; e.g., `male(albert)` is re-coded as `(male albert)`. The three-place predicate `parents` is then re-coded ("curried") into a one-place predicate `parent`, with `((parent C) P)` meaning that the parent of `C` is `P`; this also requires that each parent be specified by a separate rule, resulting in four `parent` rules instead of two.

Figure 7 illustrates the derivation of a few propositional trees from this set, using another attractor from our image gallery. The figure shows a portion of the unit square which contains the attractor (tree `X`), as well as the regions encoding the trees `(X X)` and `((X X) X)`. Sample encodings for the trees `(male edward)` and `((parent edward) victoria)` are also shown. The figure was generated as follows: First, we computed the depth-two attractor and tree-regions using the method shown in Figures 4 - 6. Then we hand-traced a closed curve in the region encoding the tree `((X X) X)`. Using a program which plots the left and right copies of the point at the current cursor position, this trace produced a left copy of this closed curve in the region encoding `(X X)`, and a right copy in the attractor region, encoding a terminal. We labeled this terminal attractor piece `victoria`. Hand tracing over the closed curve for `(X X)` resulted in a left and right copy of that curve on the attractor; we labeled these terminals `parent` and `edward` respectively. These labelings yielded a set of attractor regions that unified to the tree `((parent edward) victoria)`.

To encoded the tree `(male edward)`, we hand-traced a close curve in the region encoding `(X X)`, producing a left and right copy on the attractor. We labeled the left copy `male`and the right copy `edward'`. By taking the encoding of `edward` to be a region including both this `edward'` and the encoding of `edward` from the other tree, we obtained a set of labels unifying to both propositions. Though this is a long way from a real solution to the tree-labeling problem, it is a first step toward adding "meaning" to the abstract structural configurations we have been presenting so far.

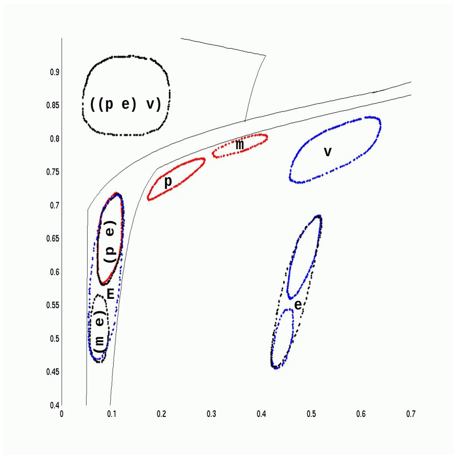

Figure 7: Encoding a few propositions using an IRAAM by hand-labeling of attractor regions. Funnel-shaped region at left side of figure encodes trees of the form `((X X) X)`. Thin diagonal band encodes trees `(X X)`. Lower-right area is the attractor. Abbreviations: `v = victoria`, `e = edward`, `p = parent`, `m = male`. Region `E` represents the set of facts about Edward: that he is male, that someone is his parent.

Conclusions and Future Work

This paper presented recent work on Infinite RAAM (IRAAM), a new connectionist architecture that fuses recurrent neural networks with fractal geometry, allowing us to understand the behavior of these networks as iterated function systems (IFSes). We have shown how limiting the number of IFS iterations allows us to use IRAAM as a connectionist substrate for unification, an algorithm that has come to play a central role in a variety of cognitive science disciplines. Fractal representation of language is a relatively new field, and we have yet to test the model on empirical data. We are however encouraged by the success of related work in fractal encoding of grammars (Tabor 2000), and see our work as contributing to this effort. We hope that such work will serve as a foundation for a principled "unification" of connectionist approaches with more traditional symbolic models, perhaps as an alternative to hybrid methods.

We see several possible research directions for our model. First, we need to apply unification-based IRAAM to something grander than a simple six-sentence database like the one used in the example. The obvious next step would be to find a non-trivial database of dozens or hundreds of propositions to test the model.

Such an effort would require a learning algorithm for IRAAM, which, given a set of propositions or other hierarchical structures, would use gradient descent or a similar method to learn a set of weights encoding just those propositions. Such an algorithm would have an error function assigning a penalty for both missing encodings and for encodings inconsistent with the examples from the training set (e.g., the proposition `female(albert)` in our example data set). An intriguing possibility would be to co-evolve both the network weights and a separate labeling program, using the paradigm described in (Hillis 1992) and (Juillé and Pollack 1996).

Finally, we suspect that inherent limitations in using a single set of network weights may hinder our attempts to use IRAAM as a model of unification in natural language, where the combinatorial possibilities are much richer than those of artificial languages like FOPC and Prolog. Research in image compression (Barnsley and Jacquin 1988) has shown the usefulness of combining several different IFSes to encode a single real-world image. We hope that a similar approach will allow IRAAM to scale up to the larger, more complicated phrases and sentences of natural language.

References

Ackley, D. H., G. Hinton, and T. Sejnowski (1985). A learning algorithm for boltzmann machines. *Cognitive Science 9*, 147–169.

Barnsley, M. F. (1993). *Fractals everywhere*. New York: Academic Press.

Barnsley, M. F. and A. Jacquin (1988). Application of recurrent iterated function systems to images. In *Proc. SPIE*, Volume 1001, pp. 122–131.

Blank, D., L. Meeden, and J. Marshall (1991). Exploring the symbolic/subsymbolic continuum: A case study of raam. Technical Report TR332, Computer Science Department, University of Indiana.

Chalmers, D. (1990). Syntactic transformations on distributed represenations. *Connection Science 2*, 53–62.

Chomsky, N. (1957). *Syntactic Structures*. The Hague: Mouton.

Clocksin, W. and C. Mellish (1994). *Programming in Prolog*. Berlin: Springer Verlag.

Dawkins, R. (1986). *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design*. New York: W.W. Norton and Co.

Fodor, J. (1975). *The Language of Thought*. New York: Crowell.

Fodor, J. and Z. Pylyshyn (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition 28*, 3–71.

Hillis, W. (1992). Co-evolving parasites improves simulated evolution as an optimization procedure. In C. Langton, C. Taylor, and J. Farmer (Eds.), *Artificial Life II*, pp. 313–324. Addison Wesley.

Horgan, T. and J. Tienson (1989). Representations without rules. *Philosophical Topics XVII*(1), 147–175.

Juillé, H. and J. Pollack (1996). Co-evolving intertwined spirals. In *Proceedings of the Fifth Annual Conference on Evolutionary Programming*. MIT Press.

Melnik, O. (2000). *Representation of Information in Neural Networks*. Ph. D. thesis, Brandeis University.

Melnik, O., S. Levy, and J. Pollack (2000). Raam for an infinite context-free language. In *IJCNN 2000*. International Joint Conference on Neural Networks: IEEE.

Pinker, S. and A. Prince (1988). On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. *Cognition 28*, 73–193.

Pollack, J. (1990). Recursive distributed representations. *Artifical Intelligence 36*, 77–105.

Rich, E. and K. Knight (1991). *Artificial Intelligence*. New York: McGraw-Hill.

Robinson, J. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM 12*(1), 23–41.

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

Rumelhart, D., G. Hinton, and R. Williams (1986). Learning internal representation by error propagation. In D. Rumelhart and J. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1. MIT.

Rumelhart, D. and J. McClelland (1986). On learning the past tenses of english verbs. In D. Rumelhart and J. McClelland (Eds.), *op. cit.*, Volume 2.

Shieber, S. (1986). *An Introduction to Unification Based Approaches to Grammar*. Number 4 in CSLI Lecture Notes. University of Chicago Press.

Siegelmann, H. (1995). Computation beyond the turing limit. *Science 268*, 545–548.

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.

Van Gelder, T. (1990). Compositionality: a connectionist variation on a classical theme. *Cognitive Science 14*, 355–384.

Williams, R. and D. Zipser (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation 1*, 270–280.