RAAM for Infinite Context-Free Languages

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

With its ability to represent variable sized trees in fixed width patterns,RAAM is a bridge between connectionist and symbolic systems. In the past, due to limitations in our understanding, its development plateaued. By examining RAAM from a dynamical systems perspective we overcome most of the problems that previously plagued it. In fact, using a dynamical systems analysis we can now prove that not only is RAAM capable of generating parts of a context free language (a^nb^n) but is capable of expressing the whole language.

1 Introduction

Recursive Auto-Associative Memory or RAAM [8] 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 a fixed-width 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 with time-varying inputs. If the training is successful, the result of bottom up encoding will coincide with top down decoding.

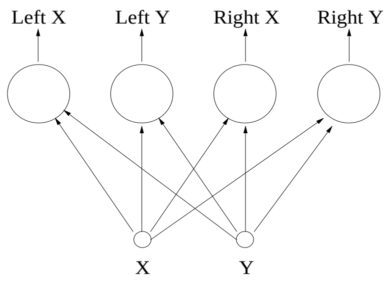
Although RAAM has found wide use in demonstration and feasibility studies, and in philosophical discussions of what could be done with networks using symbolic representations, our understanding of what RAAM could do and HOW it works has been incomplete. Depending on some factors like the trees themselves and the dimensionality of the network, learning parameters, etc., the system might or might not converge, and even when it converged it might not be reliable. Several studies of sequential RAAM demonstrated that the network could find variable-valued codings which were understandable [2]. Douglas Moreland discovered that under certain conditions sequential RAAM had a very high counting capacity [7] similar to subsequent work by [10].

The decoder works in conjunction with a logical "terminal test", which answers whether or not a given representation requires further decoding. The default terminal test merely asks if all elements in a given code are boolean, e.g. above 0.8 or below 0.2. This analog-to-binary conversion was a standard interface in back-propagation research of the late 1980's to calculate binary functions from real valued neurons. However, although it enabled the initial discovery of RAAM training, it led to several basic logical problems which prevented the scaling up of RAAM: 1) The "Infinite Loop" problem is that there are representations which "break" the decoder by never terminating. In other words, some trees appear "infinitely large" simply because their components never pass the terminal test. This behavior breaks computer program implementations or requires depth checking. 2) The "Precision vs. Capacity" problem is that tighter tolerances lead to more decoding errors instead of a greater set of reliable representions. 3) The "Terminating Non-Terminal" problem arises when there is a "fusion" between a non-terminal and a terminal, such that the decoding of an encoded tree terminates abruptly.

Various people have noticed problems in the default terminal test of RAAM and have come up with alternatives, such as simply testing membership in a list, or simultaneously training a "terminal test network" which classifies representations as terminal or nonterminal. [3]. In addition, there are attempts at increasing capacity through modularization and serialization [5, 11].

In the rest of this paper we present a new formulation of RAAM decoders based on an analysis of the iterated dynamics of decoding, that resolves all these problems completely. This formulation leads to a new "natural terminal test", a natural labeling of terminals, and an inherent higher storage capacity. We then continue the dynamical systems analysis to prove that based on a prototype a^nb^n RAAM decoder generated by hill-climbing, we can generate a competence class of parameterized decoders that not only exclusively generate a^nb^n sequences but in some cases are capable of generating the full infinite language.

2 New RAAM Formulation



$$\begin{aligned} Left_X &= \frac{1}{1+e^{-(w_{LXX}x+w_{LXY}y+w_{LX})}} \\ Left_Y &= \frac{1}{1+e^{-(w_{LYX}x+w_{LYY}y+w_{LY})}} \\ Right_X &= \frac{1}{1+e^{-(w_{RXX}x+w_{RXY}y+w_{RX})}} \\ Right_Y &= \frac{1}{1+e^{-(w_{RYX}x+w_{RYY}y+w_{RY})}} \end{aligned}$$

Figure 1: An example RAAM decoder that is a 4 neuron network, parameterized by 12 weights. Each application of the decoder converts an (X, Y) coordinate into two new coordinates.

Consider the RAAM decoder shown in figure 1. It consists of four neurons that each receive the same (X, Y) input. The output portion of the network is divided into the right and left pairs 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) [1], consisting of two pseudo-contractive transforms which are iteratively applied to points in a two dimensional space.

In the past we have examined the applicability of the IFS analogy to other interpretations of neural dynamics [9, 4, 6]. But in the context of RAAMs the main interesting property of contractive IFSes lies in the trajectories of points in the space– when we take a point and recursively apply the transforms to it (applying both or randomly choosing between them) where will the point eventually end up? For contractive IFSes the space is divided into two sets of points: The first set consists of points located on the underlying attractor (fractal attractor) of the IFS. Points on the attractor never leave it. That is, repeated applications of the transforms to points on the attractor only moves them within the attractor– effectively coercing them to take on orbits within its confines. The second set is the inverse 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. Finite, multiple iterations of the transforms have the effect of bringing the points in this second set arbitrarily close to the attractor.

As noted before, problems 1 and 3 arise from a problematic terminal test. To solve them, there needs to be a clear separation between terminals and non-terminals, such that for any non-terminal starting point a terminal test must eventually "catch" a trajectory delineated by the decoder's dynamics. Since some trajectories never leave the attractor and all others eventually hit the attractor, a terminal test must contain points on the attractor. There is no gurantee that a trajectory on the attractor will hit all parts of the attractor, therefore the only terminal test that guranttees the termination of all trajectories of the RAAM (IFS) is a test that includes all the points of the attractor itself.

By taking the terminal test of the decoder network to be "on the attractor", not only are problems of infinite loops and early termination corrected, but it is now possible to have extremely large sets of trees represented in small fixed-dimensional neural codes. The attractor, being a fractal, can be generated at arbitrary resolution (see [1] or [6] for how the attractor is generated). 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. For this formulation, the set of trees generated and represented by a specific RAAM are a function of the weights, but are also governed by how the initial condition space is sampled, and by the resolution of the attractor construction. Note that the lower resolution attractors contain all the points of their higher dimensional counterparts (they cover them), therefore as a coarser terminal set they terminate trajectories earlier and therefore act to "prefix" the trees of the higher dimensional attractors.

Two last pieces complete the new formulation. First, the encoder network, rather than being trained, is constructed directly as the mathematical inverse of the decoder. The terminal set of each leaf of a tree is run through the inverse left or right transforms, and then the resultant sets are intersected and any terminals subtracted. This process is continued from the bottom up until there is an empty set, or we find the set of initial conditions which encode the desired tree.

Second, using the attractor as a terminal test also allows a natural formulation of assigning labels to terminals. Barnsley [1] 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 the attractor 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 sufficent number of significant digits from this address.

3 Hill-Climbing an $a^n b^n$ decoder

We used hill-climbing to arrive at a set of RAAM decoder weights for the simple non-regular context-free language $a^n b^n$; that is, the set of strings consisting of a sequence of a's followed by an equal-length sequence of b's. Two ways to represent the targets for hill-climbing would be either a set of strings in $a^n b^n$: *ab*, *aabb*, *aaabbb*, ..., or a set of parenthesized expressions representing binary-branching trees having those strings at their frontiers: (ab), $((a(ab))b)$, $((a((a(ab))b))b)$, ... RAAM is a method for representing structure, and not just strings of symbols. Therefore, we chose the latter, tree-based representation. Specifically, we used trees generated by a simple context free grammer which generates $a^n b^n$. We were guided by the assumption that this choice would drastically restrict the set of possible solutions to be explored and allow our hill-climbing RAAM to build upon existing structure as it navigated the space of decoder weights.

For the hill-climbing, both the initial random weights and the random noise added to each weight came from a Gaussian distribution with zero mean and a standard deviation of 5.0. Starting with 12 random decoder weights, we explored the space of weights by adding random noise to each weight and using the resulting weights to generate trees on a 64-by- 64 fractal RAAM. That is, the attractor was generated at that resolution and the initial starting point space was also sampled at that resolution. The terminals of these trees were addressed with an a or a b, using the scheme described in the section above. We used 10 trees representing strings from $a^n b^n$ and $a^{n+1} b^n$ with n = 1, 2, 3, 4, and 5 which had subpart relationships (e.g., the tree for $a^2 b^2$ is a subpart of the $a^3 b^3$ tree) for our learning set.

About a third of the trials were able to mutate successfully into patterns that "covered" the training set, yielding all ten tree structures, as well as trees of the form $a^n b^{n+1}$, plus additional, ill-formed trees. Though we were able to generate many different weight set solutions to cover the training data, figure 2 shows that all the solutions had a dramatic "striping" pattern of tree equivalence classes, in which members of a single class were located in bands across the unit square. (Recall that any point not on the attractor represents a tree.) So, for example, the wide gray band occupying most of the top of the image at right represents the equivalence class for the tree (ab). Furthermore (and less noticeable in the figure), the attractor for these hill-climbed weights was located on or toward the edge of the unit square. In the figure below, the b attractor points are the white squares on the right side of the image at left.

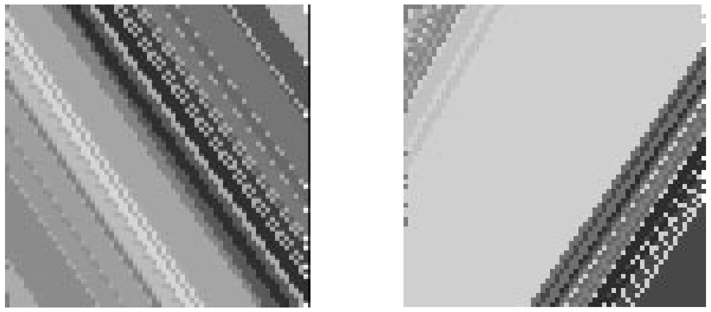


Figure 2: The equivalence classes of two solutions to $a^n b^n$ found by hill-climbing.

Beyond the 64-by-64 resolution for training the RAAM did not generalize deeply. However, the dramatic consistency in the solution patterns led us to wonder whether there was an underlying formal solution toward which our $a^n b^n$ hill-climbing RAAM was striving. As we discuss in the next section, the answer to this question turned out to be positive.

4 Competence model and proof

We claim that the RAAM evolved by our hill-climbing experiment is indicitive of a class of RAAM competence models which generate a^nb^{n+1} and $a^{n+1}b^n$ languages. We justify our claim by demonstrating how an analysis of the specific RAAM dynamics garners the principles to design a parameterized class ofa^nb^{n+1} and $a^{n+1}b^n$ RAAMs, some of which, in the infinite case generate the whole languages.

The dynamics of our RAAM can be examined by an arrow diagram. Starting from an initial point, we apply both transformations, and plot arrows to designate the new points thus generated. This process is continued until all new points are on the terminal set.

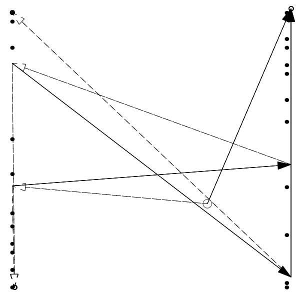


Figure 3: An arrow diagram of a tree starting at (0.7,0.3).

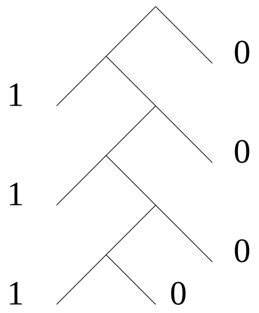


Figure 4: A typical tree generated by a zigzag RAAM.

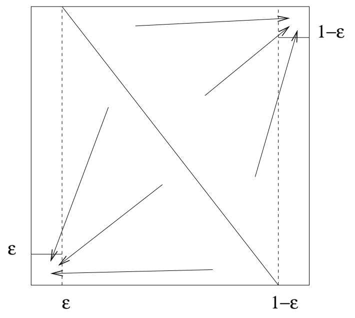


Figure 5: The line that divides the space of points that go to the upper right and lower left

In figure 3 we see a typical example of an arrow diagram from our evolved RAAM. The initial point is marked by a circle. The terminal points on the left side are the 0 terminals and the terminals on the right side are the 1 terminals. The solid line corresponds to the right transform, and the dashed lines correspond to the left transform. By examining the dynamics we can discern a few specific properties:

1. The top right and bottom left corners are both terminals. From any initial starting point, at least one of the transforms goes to a corner terminal.
2. The left transform always takes us to the left side. The right transform always takes us to the right side.
3. On the left side, the left transform takes us to the bottom left corner terminal. On the right side the right transform takes us to the upper right corner terminal.

The effect of these properties is to generate a very specific variety of trees. By property 1, we see that from any initial point our tree will immediately have one of its branches terminate. By property 2 we see that the continuing branch will hit one of the sides. But by property 3 we see that this branch will also lose one of its sub-branches, only continuing the tree across one branch.

We can characterize this behavior as follows: The tree dynamics consist of a zigzag line which goes between the left side and right side. At each side one of the transforms goes to a terminal, while the other continues the zigzag. This continues until the zigzag hits a terminal on one of the sides.

The effect of these zigzag dynamics is that at each successive tree level we get an alternating 0 or 1. In figure 4 we see what such a tree might look like.

It is apparent that any set of transforms which obey properties 1,2 and 3 will generate trees of this sort. In order to demonstrate the existence of a competence model we need to show that such transforms exist for any resolution, and that they can generate arbitrarily sized trees.

Let $\varepsilon > 0$ be the width of a terminal point. Properties 1 and 2 mandate a parameter dependency on ε . This is due to the transform being composed of a sigmoid function, hence it can never quite reach 0 or 1, but can be made arbitrarily small, so no single transform can fulfill properties 1 and 2 for any ε.

By property 3, in the rectangular region defined by the coordinates $(0,0) \times (\varepsilon, 1)$ the left transform goes to the bottom left corner $(x, y \le \varepsilon)$ and in the rectangular region $(1-\varepsilon, 0) \times (1,1)$ the right transform goes to the upper right corner ($x, y \geq 1-\varepsilon$). Therefore to fulfill property 1 as well we can divide the space into two regions, defined by the line that connects $(\varepsilon, 1)$ and $(1-\varepsilon, 0)$. On the upper half of the line the right transform will go to the upper right corner, and on the bottom half of the line the left transform will go to the bottom left corner. See figure 5.

The equations that define the RAAM are given in figure 1. Since the X-transforms always take us to their respective sides, we can make them constant by setting $w_{LXX} = w_{LXY} = w_{RXX} = w_{RXY} = 0$, $w_{LX} = -\log\frac{1-\varepsilon}{\varepsilon}$ and $w_{RX} = -\log\frac{\varepsilon}{1-\varepsilon}$. Thus the left x-transform will always take us to ε and the right x-transform will always take us to $1-\varepsilon$.

The line from $(\varepsilon, 1)$ to $(1-\varepsilon, 0)$ can be parameterized by $cx + c(1-2\varepsilon)y - c(1-\varepsilon) = 0$, where c is a constant. If we set $c > 0$, then we can plug these values directly into our transforms. We need to set $w_{LYX} = w_{RYX} = c$, $w_{LYY} = w_{RYY} = c(1-2\varepsilon)$ and to adjust the constants to $w_{LY} = -c(1-\varepsilon) - \log\frac{1-\varepsilon}{\varepsilon}$ and $w_{RY} = -c(1-\varepsilon)) - \log\frac{\varepsilon}{1-\varepsilon}$. These weights guarantee the right transform takes all points above the line to the upper right terminal and vice-versa for the left transform. These weight are dependent on ε, thus we have demonstrated the existence of weights which obey all three properties for any resolution.

To show the existence of arbitrarily sized trees we need to examine the terminal set locations and the specific dynamics of the zigzag line. In figure 6 you see an arrow diagram for the points on the terminal set for a particular setting of c and ε. This diagram shows what points on the attractor go to what other points on the attractor. We see a few interesting characteristics of the terminal set: Initially, we see that out of the two terminal corners we get a zigzag line which seems to generate the terminal set points for the upper and lower part of the space. Both of these zigzags terminate before intersecting. Since these zigzags always go to the corners on their respective sides, they are in fact orbits of different lengths which include a corner. The only other terminals are a pair of points somewhere in between the termination of the two zigzags. These points just go back and forth between themselves.

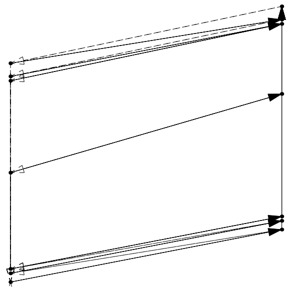


Figure 6: A terminal point arrow diagram for $\varepsilon = 1/64$ and $c = 5.703$.

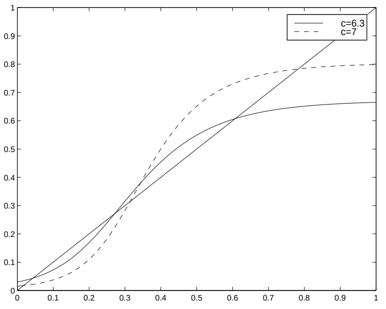


Figure 7: A one-dimensional map diagram of the zigzag line for two values of c.

We can analyze these different properties by treating the zigzag as a one-dimensional map. From every point on the left side the application of the right and then left transform brings us to a new point on the left side. In figure 7 we see two one dimensional maps which both have an ε value of 1/256 but two values of c, 6.3 and 7. By drawing the line of slope 1 through these maps we see that they have 3 fixed points. Two of these are stable, attractive fixed points (this can be shown by linearizing about the fixed points). We can see that the attractor zigzags which emanate out of the corner terminals go towards these fixed points and terminate on them. The third fixed point in the center, is unstable, so zigzag trees above it will head towards the upper half of the space and zigzags below it will head towards the bottom half of the space.

As can be seen in 7 the location of this unstable fixed point is dependent on c. Since this dependence is continuous, and between $c = 6.3$ and 7 the location of the fixed point moved by more than 1/256, then by the mean value theorem we know that there exists a c such that the location of the unstable fixed point is at an integer multiple of ε . Assume that we are discretizing by flooring to the nearest integer multiple of ε, then since the unstable fixed point is at an integer multiple of ε, we can pick an intial tree starting point on the left side, below it, which is arbitrarily close to it. The unstable fixed point represents an orbit which never reaches the other terminal points. If it were not a terminal it would correspond to an infinite tree. By coming from below it we are guranteed not to hit it. But by coming arbitrarily close to it we can generate arbitrarily long zigzag trees, which hover in its vicinity for an arbitrary number of iterations before heading towards the attractive fixed point below it. Thus we can create trees of any size, and this model can generate the whole language in an infinite resolution sense.

5 Discussion

By examining the underlying dynamics, we have presented a new approach to understanding RAAM. This approach allows us to describe the "natural terminal test" of a RAAM decoder as well as represent trees in accordance with the intrinsic dynamics. The most serious problems of the original RAAM are solved by this new reformulation, allowing a RAAM decoder to provably terminate for any input and potentially generate an infinite grammar.

Specifically, based on initial weights generated by a evolved decoder, we have generalized and proven that there exist a set of 12 weights for a RAAM decoder which not only exclusively generate words from the $a^{n(+1)}b^{n(+1)}$ language, but are capable of generating the whole language. In fact in the region of the unstable fixed point, the RAAM exhibits a monotonic behavior. As we approach the fixed point, the size of the generated trees increases divergently. The significance of this is not only in the context of RAAM but in the context of connectionist processing in general, since we have demonstrated how a monotonically increasing continuously varying input (initial starting point) can incrementally generate the members of a context free grammar, within the confines of a neural substrate. Thus we have shown how a smooth mapping could exist between the tonicly varying outputs of a single neuron and the generativity of a context free grammar.

With this reformulation of RAAM, we have a new and deeper connection between connectionist and symbolic representational and generative theories. In the future we expect to examine the fuller potential of this model, including the expressiveness of other grammars and modes of lexical assignment, as well as more informed learning methods.

[1] M.F. Barnsley. *Fractals everywhere.* Academic Press, New York, 1993.

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

[3] L. Chrisman. Learning recursive distributed representations for holistic computation. *Connection Science*, 3(4):354–366, 1991.

[4] J.F. Kolen. *Exploring the Computational Capabilities of Recurrent Neural Networks.* PhD thesis, Ohio State, 1994.

[5] S.C. Kwasny, B.L. Kalman, and A. Abella. Decomposing input patterns to facilitate training. In *Proceedings of the World Congress on Neural Networks*, volume 3, pages 503–506, Portland, Oregon, 1993.

[6] O. Melnik and J.B. Pollack. A gradient descent method for a neural fractal memory. In *WCCI 98.* International Joint Conference on Neural Networks, IEEE, 1998.

[7] D.D. Moreland. An investigation of input encodings for recursive auto associative neural networks. Master's thesis, Ohio-State, 1989.

[8] J.B. Pollack. Recursive distributed representations. *Artifical Intelligence*, 36:77–105, 1990.

[9] J.B. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7:227–252, 1991.

[10] P. Rodriguez, J. Wiles, and J.L. Elman. A recurrent neural network that learns to count. *Connection Science*, 11:5–40, 1999.

[11] A. Sperduti and A. Starita. An example of neural code: Neural trees implemented by lraam. In *International Conference on Neural Networks and Genetic Algorithms*, pages 33–39, Innsbruck, 1993.