A Gradient Descent Method for a Neural Fractal Memory

Ofer Melnik and Jordan Pollack, Volen Center for Complex Systems, Brandeis University, Waltham, MA, USA

Abstract—
It has been demonstrated that higher order recurrent neural networks exhibit an underlying fractal attractor as an artifact of their dynamics. These fractal attractors offer a very efficent mechanism to encode visual memories in a neural substrate, since even a simple twelve weight network can encode a very large set of different images.

The main problem in this memory model, which so far has remained unaddressed, is how to train the networks to learn these different attractors. Following other neural training methods this paper proposes a Gradient Descent method to learn the attractors. The method is based on an error function which examines the effects of the current network transform on the desired fractal attractor. It is tested across a bank of different target fractal attractors and at different noise levels. The results show positive performance across three error measures.

Keywords— **Recurrent Neural Networks, Dynamical Systems, Fractals, Iterated Function Systems, Inverse Fractal Problem, Learning Rules, Gradient Descent.**

I. Introduction

THERE have been numerous interpretations of the function that dynamics serve in recurrent neural networks. The Hopfield network uses the fixed points of the network dynamics to represent memory elements. Networks studied by Pollack [1], Giles [2], and Casey [3] use the current activation of the network as a state in a state machine while using the dynamics of the network as the transition map. Some try to model existing dynamical systems with recurrent neural networks [4]. RAAMs [5] use the network dynamics to describe complex data structures such as trees and lists.

We employ a different interpretation of network dynamics [6], in which the network is treated as an *Iterated Function System* that is coding for its underlying fractal attractor [7]. The fractal attractors used in this work are two dimensional, hence the network is in effect coding for fractal images, and may be acting as a form of visual memory.

Iterated Function Systems (IFS's) are a set of simple functions. Each function receives as input a coordinate from a space and returns a new coordinate which is usually a simple transformation of the input coordinate. When these functions are applied iteratively to points in a space, they converge on a set of points, called the IFS's attractor. This attractor is a fractal, a set with similar structure at different resolutions.

The connection between IFS's and recurrent neural networks comes from thinking of a network's neurons as the functions or transforms of an IFS. As such, these neurons receive (X,Y) coordinates as input and return new ones in a recurrent manner.

It has been suggested that coding fractals by Iterated Function Systems may be an effective mechanism for compressing images [8]. As such this interpretation of network dynamics may form the basis of a highly efficient method for storing visual information and other related memories.

A small sample of some of the fractals which a simple network of only four neurons can encode is shown in Figure 1. It is conceivable that this rich and interesting set of fractals may be used to encode real-world visual images or at least some of their properties.

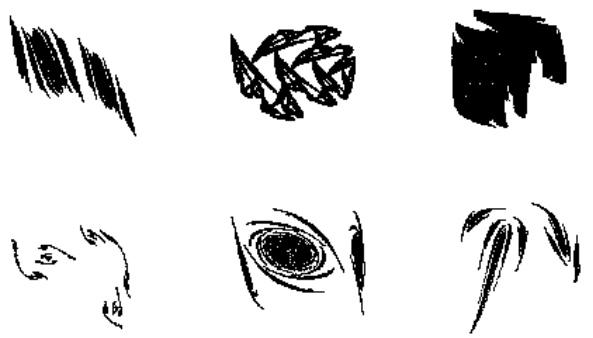

Fig. 1. A sample of fractal attractors which can be generated with a four neuron neural network.

To make this interpretation of recurrent neural networks as storing a fractal attractor applicable, it is necessary to demonstrate a mechanism by which these attractors can be learned or approximated by the network. This issue is related to the *Inverse Fractal Problem*, an area of active research, which asks how, given an image, do you find the Iterated Function System which can generate the image.

There have been different approaches towards solving the inverse fractal problem; the main motivation for this research has been image compression using fractals. Many of the approaches use generalizations of IFS's such as LIFS's which use local-similarity as well as self-similarity. The *method of moments* uses invariant measures of moments to match a function system to a target image [9]. *Genetic Algorithms* have also been used to address this problem [10]. The current generation of successful fractal image compression algorithms succeed by severely limiting the space of transforms to be used [11], [12]. At present there is still no general algorithm for solving the inverse fractal problem. It seems to be an elusive problem related to the classic problem of object recognition under transformation.

In the vein of other learning algorithms for neural networks, such as the ubiquitous *backprop* [13] and many which have come since, we developed a training method for our network which relies on an energy or error function that we seek to minimize by means of a gradient descent on its energy landscape. In effect, minimization of this error function will lead to the network learning the desired attractor. In the rest of this article we describe the network architecture employed, explain the ideas behind the choice of the error function and evaluate its efficacy.

II. Architecture

A neural network based IFS can have an arbitrary number of transforms. We have constrained ourselves to two transforms in this implementation, because this is the minimal amount needed to have a rich set of underlying attractors.

Since each transform is a mapping from one X,Y coordinate to another, it must be composed of two scalar functions - one function for each output component, either X or Y. Therefore the recurrent network we employed in our study consists of four neurons, operating as a two transform IFS, where all neurons receive an X,Y coordinate as input and return either the X or Y component of a transform.

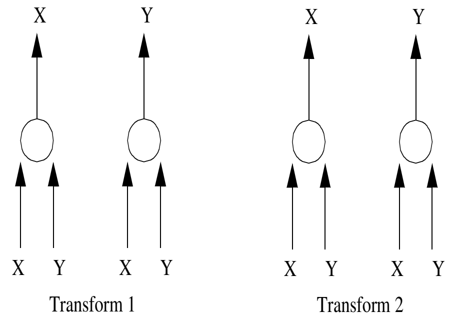

Fig. 2. The neural network architecture consists of four neurons each pair acting as a transform.

The function each neuron computes is the standard sigmoid of the weighted sum of the inputs, with a bias term:

$$Out = \frac{1}{1 + e^{-w_x X + w_y Y + w_1}}$$

As such each neuron has three modifiable parameters giving a total of twelve parameters for the whole network.

Since the attractor is an intrinsic property of its IFS, different methods can be used to generate it. In our implementation in order to generate the fractal attractor at a user specified resolution, we initially located a single point on the attractor, by applying one of the transforms on a random point for a number of steps, until it converged. Next, the network was iteratively run on this point, generating new points on the attractor. Then the process was repeated for all new points, until no new points were found, and the set of points on the attractor was assumed to be complete.

III. Learning

The problem we are trying to solve is: given a fractal attractor, find a set of weights for the network which will approximate the attractor. Our approach to this problem consists of finding an error function which will be minimized when the *network coded attractor* is equal to the *desired attractor*.

A common metric or error function used to compare fractal attractors is the Hausdorff distance [7]. The distance is calculated by finding the farthest point on each set relative to the other set and returning the maximum of these two distances. By calculating from both sets in a symmetrical manner, the Hausdorff distance gives a measure of mutual overlap. In other words it will equal zero only when each set is contained within the other, or when they are both equal. The Hausdorff distance between two point sets, A and B, is defined as:

$$H(A, B) = \max\left(h, (A, B), h(B, A)\right)$$

where

$$\begin{aligned} h(A, B) &= \max\{\Delta(a, B) \mid a \in A\} \\ \Delta(a, B) &= \min\{\|b - a\| \mid b \in B\} \end{aligned}$$

The Hausdorff distance does not lend itself to a gradient descent approach to minimization because it is not differentiable. In the fractal case, this is due to two reasons. First, the fractal attractor is generated by an iterative process, which is inherently not differentiable. Second, the Hausdorff distance effectively uses only one point from each set. This point is not constant and its selection may lead to discontinuities.

Our error function borrows principles from the Hausdorff distance and the *collage thereom* [8]. The collage thereom provides the basis for most approaches to the fractal inverse problem or fractal image compression. It states that in order to find an IFS for a given fractal attractor, it is necessary to find a transformation which maps the attractor to itself. As such, our error function will be minimized when the desired attractor and *transformed desired attractor* mutually overlap. Since the network transformations are pseudo contractive due to the sigmoid non-linearity, it follows from the collage thereom that this is equivalent to the network coded attractor being equal to the desired attractor.

As stated previously, there are two issues which need to be addressed to make our error function differentiable: the issue of attractor generation being iterative and the issue of only using one point to calculate the error function. By comparing the desired attractor with the transformed desired attractor instead of the network coded attractor as per the collage theorem, we can overcome the iterative process issue. This works since, instead of comparing two attractors generated by an iterative process, we will be comparing one attractor with the same attractor acted on by a function, one iteration of the IFS. The second differentiabilty issue is the number of points used in actually calculating the error function. Our approach is to sum over all points, both on the desired attractor and transformed desired attractor, rather than selecting only the furthest points.

For a given attractor A and a set of transforms T our error function is defined as follows.

$$E(T,A) = \sum_i \sum_{x,y\in A} \Delta\left(T_{ix}(x,y), T_{iy}(x,y), A\right) + \sum_{x,y\in A} \Delta\left(x,y,T(A)\right)$$

Where T_{ix} and T_{iy} represent the i-th transform for x and y respectively (as calculated by each neuron) and $T(A)$ is the transformed attractor. The definition for Δ here is:

$$\Delta(x,y,A) = \min\left\{\|(x,y)-a\|^2 \mid a \in A\right\}$$

This error function is similar to the Hausdorff metric in being symmetrical, i.e., taking distances from the desired attractor to the transformed desired attractor as well as distances from the transformed desired attractor to the desired attractor. It is also similar in its use of the Δ function for measuring the distance between a point and a set. There are two advantages to this error function: first, by summing over all the points in the calculation, we get a better measure of the number of points of actual mutual overlap. Second, this error function is practically differentiable with respect to the weights of the transform, allowing its use in our gradient descent approach to minimization.

In examining the error function, it is apparent that it is composed of essentially continuous and differentiable functions. The only part which is not is the min function. In most applications the min function is not continuous, but in this particular case it is. This continuity stems from the continuity of Δ with respect to its parameters. For example, if we were to examine the continuity of Δ with respect to the x variable, we can imagine that for a while the min function picks a certain point on the attractor which gives the minimum distance, and at some value of x the min function switches to another point on the attractor. However, by geometric reasoning we know that while switching to that other point there is a certain intermediate x for which the distances to the first and second points are equal. Thus there is no jump in the value of the min function when switching points, therefore it is continuous.

A similar argument can be presented for y and the weights of the transform. Continuity does not mean differentiabilty for the min function. At the points where the min function could go to multiple points on the attractor there is no single derivative. Since this is a relatively rare event we have chosen to pick arbitrarily one of the points for the derivative calculation.

Due to the fact that in the first term of E the transform applies to x and y, and in the second term it applies to the set of the attractor, the derivatives must be handled differently for each term. Specifically, for each point in $T(A)$ we must remember the point in A from which it came and which transform was used.

We now calculate the gradient of the function E with respect to the weights of a T_x transform. The calculation for the other transforms is identical and will not be repeated. The gradient is given by:

$$\left(\frac{\partial E}{\partial w1}, \frac{\partial E}{\partial w2}, \frac{\partial E}{\partial w3}\right) = \frac{\partial E}{\partial \Delta}\cdot\frac{\partial \Delta}{\partial T_x}\cdot\left(\frac{\partial T_x}{\partial w1}, \frac{\partial T_x}{\partial w2}, \frac{\partial T_x}{\partial w3}\right) + \frac{\partial E}{\partial \Delta}\cdot\frac{\partial \Delta}{\partial T(A)}\cdot\left(\frac{\partial T(A)}{\partial w1}, \frac{\partial T(A)}{\partial w2}, \frac{\partial T(A)}{\partial w3}\right)$$

The derivative for the transform is the standard one used in backprop as given by:

$$\left(\frac{\partial T_x}{\partial w1}, \frac{\partial T_x}{\partial w2}, \frac{\partial T_x}{\partial w3}\right) = x_a(1-x_a)(x,y,1)$$

Therefore the gradient can be written as

$$\left(\frac{\partial E}{\partial w1}, \frac{\partial E}{\partial w2}, \frac{\partial E}{\partial w3}\right) = \sum_{x,y\in A} 2(T_x - x_{\Delta 1})T_x(1-T_x)(x,y,1) + \sum_{x,y\in A} 2(x - x_{\Delta 2})x_{\Delta 2}(1 - x_{\Delta 2})(x^o_{\Delta 2}, y^o_{\Delta 2}, 1)$$

where T_x refers to $T_x(x,y)$; $x_{\Delta 1}$ is the x chosen by the min function of the Δ function in the first term of E ; $x_{\Delta 2}$ is the x chosen by the min function of the Δ function in the second term of E ; and $x^o_{\Delta 2}, y^o_{\Delta 2}$ are defined by $x_{\Delta 2} = T_x(x^o_{\Delta 2}, y^o_{\Delta 2})$.

Notice that for the second term $x_{\Delta 2}$ may not exist with respect to the mapping from the attractor given by a particular T_x, or be mapped to multiple times. By adjusting the sum respectively, both terms in E even out in magnitude.

IV. Evaluation and Discussion

The motivation behind the evaluation of the error function was twofold. First, we wanted to demonstrate that fractal attractors are learnable using this error function. Second, we wanted to assess the domain in which the error function was successful. This was done by applying the algorithm across a diverse set of fractal attractors, and by varying the initial noise conditions. Since it is well known that it is difficult to objectively gauge the difference between images, we chose to use three different metrics to evaluate the results as well as our own eyes.

The error function was tested on a set of 100 fractal attractors. The fractals used were randomly selected from a set of fractals previously generated using a hill-climbing algorithm to locate non-point attractors. For each fractal the set of weights used to generate the attractor with a certain amount of noise were used as initial conditions for the gradient descent algorithm. The attractors were represented at 16 by 16 pixels. The weight values ranged between -5 to 5. The uniform noise introduced had a variance of 0.25, 0.5, 1.0 and 4.0 across different trials. The gradient descent was weighted using a linear decay function, and consisted of 20 iterations. For each noise level 10, trials were conducted on each attractor, thus 4000 trials were conducted.

The performance of the algorithm was gauged using the Hausdorff distance, Hamming distance, and similar to the error function described in this article, the sum of the point distances between two attractors. These distances to the desired attractor were computed on the initial conditions and on the final conditions after the gradient descent run. On average, 79 out of the 100 attractors tested showed an improvement across all three measures.

In Figure 3 we see subjective confirmation of the success of the algorithm. In the example runs A, B, C and D, we see that the final network coded attractor is very similar to the desired attractor, in spite of having a different initial network coded attractor. Sample runs E and F show how the algorithm may fail, since we see that the final network coded attractor differs more from the desired attractor than the initial network coded attractor.

It seems that the algorithm performs more impressively with higher initial error. This is seen clearly in Figure 4, where a histogram of changes in the Hausdorff distance across trials is displayed. We see that for the 4.0 and 1.0 error cases, across most trials, there is a strong negative change in the Hausdorff distance, meaning an improvment. However for the 0.5 and 0.25 error cases, the changes do not seem as significant and unilateral. One obvious explanation for this effect might be that a larger initial error would bring a greater perceived improvement. However this still does not account for the skew in these graphs, which may be due to the large initial jumps in the gradient descent algorithm and the non-linear nature of the error landscape.

The graphs in Figure 5 suggest that in many cases the algorithm is driving towards the global minimum, as opposed to local minima. This is seen most clearly in the error function graph, where most of the final network coded attractors have an error near zero, meaning they are close to the desired attractor. Across the other two measures this migration towards zero is present but less obvious. We would expect it be most conspicuous in the error function which is closest to the measure used in the actual gradient descent.

In physically examining the results across the different attractors, it appears that the algorithm performs better on non-overlapping transforms and spatially distinct attractors. This makes sense because there is less ambiguity with respect to the transformation in relation to the self similarity of the attractor.

This algorithm represents the first step in harnessing fractal attractors of recurrent neural networks for computation. There are still many avenues of exploration with this error function. For example, manipulating the error constant decay in ways appropriate to the degree of error, running simulations on different categories of attractors, using ensemble techniques, as well as modifying the error function itself.

These network fractal attractors can be used in different applications. Obviously they can be used to store images. But they can also represent operating ranges and domains. For example an attractor may represent the range of freedom of a joint or the field of vision from a vantage point. The main advantage of using fractal attractors is their inherently very compact coding of visual information. For this reason we believe that this approach warrants further research in the context of neural storage of visual information.

V. Acknowledgments

This paper is supported in part by NSF grant IRI-9529298 and the Sloan Center for Theoretical Neurobiology at Brandeis University.

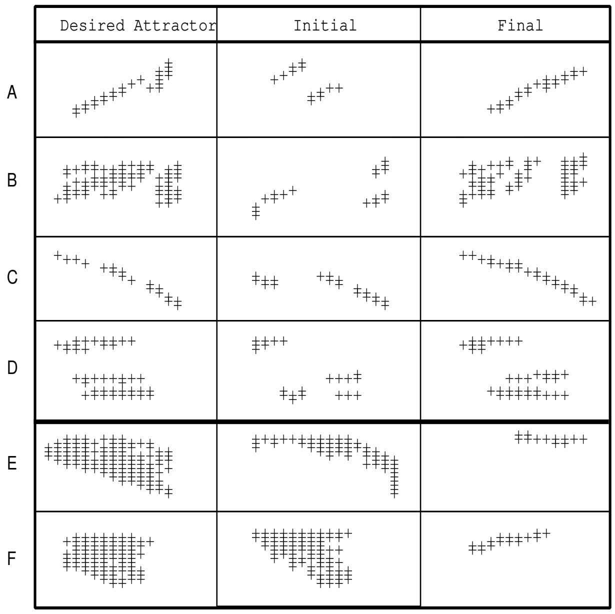

Fig. 3. This figure shows some example run results. These are 16 by 16 attractors. The first four are examples of subjective successes while the last two leave something to be desired.

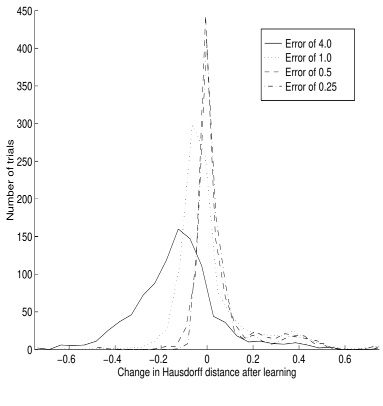

Fig. 4. This is a histogram of changes in the Hausdorff distance after running the gradient descent algorithm. It was compiled across all 4000 trials. Each of the different graphs represent trials at different initial error levels.

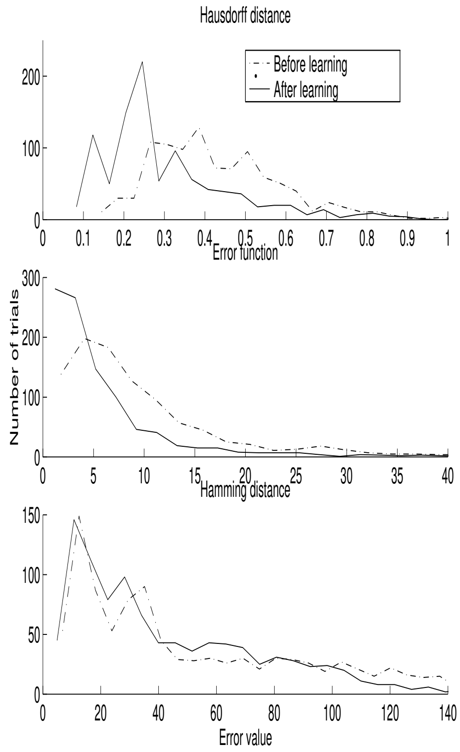

Fig. 5. Each graph represents a histogram of either the initial or final error across each of the three error measures. These were conducted for an initial error level of 4.0.

References

[1] J.B. Pollack, "The induction of dynamical recognizers," *Machine Learning*, vol. 7, pp. 227–252, 1991.

[2] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, and Y.C. Lee, "Learning and extracting finite state automata with second-order recurrent neural networks," *Neural Computation*, vol. 4, no. 3, pp. 393–405, 1992.

[3] M. Casey, "The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction," *Neural Computation*, vol. 8, no. 6, 1996.

[4] F.S. Tsung and G.W. Cottrell, "Phase-space learning for recurrent networks," Tech. Rep. CS93-285, Dept. Computer Science and Engineering, University of California, San-Diego, 1993.

[5] J.B. Pollack, "Recursive distributed representations," *Artificial Intelligence*, vol. 46, pp. 77–105, 1990.

[6] D.J. Stucki and J.B. Pollack, "Fractal (reconstructive analogue) memory," in *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*. Cognitive Science Society, 1992, pp. 118–123.

[7] M.F. Barnsley, *Fractals Everywhere*, Academic Press, San Diego, 1988.

[8] M.F. Barnsley and L.P. Hurd, *Fractal Image Compression*, AK Peters, Wellesley, 1992.

[9] E.R. Vrscay and C. J. Roehrig, *Iterated Function Systems and the Inverse Problem of Fractal Construction Using Moments*, pp. 250–259, Springer-Verlag, New York, 1989.

[10] R. Shonkwiler, F. Mendivil, and A. Deliu, "Genetic algorithms for the 1-d fractal inverse problem," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego, 1991.

[11] A. Jacquin, "A novel fractal block-coding technique for digital images," in *IEEE ICASSP Proc. 4*. IEEE, 1990, pp. 2225–2228.

[12] Y. Fisher, *Fractal Image Compression*, Springer-Verlag, New York, 1994.

[13] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning Internal Representations by Error Propagation*, vol. 1, chapter 8, MIT Press, Cambride, MA, 1986.