Pergamon

Comput. & Graphics, Vol. 22, No. 1, pp. 143–149, 1998
© 1998 Published by Elsevier Science Ltd. All rights reserved
Printed in Great Britain
PII: S0097-8493(97)00089-7
0097-8493/98 $19.00 + 0.00

Chaos & Graphics

ARTIFICIAL NEURAL NET ATTRACTORS

J. C. SPROTT

Department of Physics, University of Wisconsin, Madison, WI 53706, USA

Abstract—Aesthetically appealing patterns are produced by the dynamical behavior of artificial neural networks with randomly chosen connection strengths. These feed-forward networks have a single hidden layer of neurons and a single output, which is fed back to the input to produce a scalar time series that is always bounded and often chaotic. Sample attractors are shown and simple computer code is provided to encourage experimentation. © 1998 Published by Elsevier Science Ltd. All rights reserved

Dynamical systems modeled by nonlinear maps and flows can produce an astonishing variety of aesthetically appealing visual forms [1]. One such nonlinear map is an artificial neural network [2]. Neural networks offer a number of advantages as generators of interesting visual patterns: 1. Their outputs are automatically bounded with the proper choice of a squashing function. 2. Neural networks are universal approximators [3] and hence are capable of generating any pattern if they are sufficiently complicated. 3. There is a large literature on the design, training, and behavior of neural networks. 4. Neural networks mimic the operation of the human brain, and hence they are a natural choice for emulating human-generated art. 5. In principle, they can be trained to improve the quality of their art, just as a human can be trained.

There are many possible neural network architectures. The one used here is the feed-forward network shown in Fig. 1. It has an input layer with D elements (y_1, y_2,...,y_D) a hidden layer of N neurons (x_1, x_2,...x_N), and a single output y_0. The network is characterized by the equations:

$$x_i = \tanh\left(\sum_{j=1}^{D} w_{ij} y_i\right) \tag{1}$$

$$y_0 = s \sum_{i=1}^{N} \beta_i x_i \tag{2}$$

where w_{ij} is a matrix of connection strengths (weights), β_i is a vector of connection strengths, and s is a scaling factor, which could have been absorbed into β s. The hyperbolic tangent in Equation (1) is given by

$$\tanh(u) = 1 - 2/(e^{2u} + 1). \tag{3}$$

This squashing function is one of several that are commonly used. It is bounded in the interval (–1, 1) and anti-symmetric about the origin.

Networks of this type have been used extensively for pattern recognition [4, 5]. For example, the inputs might be the intensities of the pixels in a rectangular box containing the image of a single numerical digit (0, 1, ...9), and the output could be an analog signal close to the value of the digit. Such a network would have to be trained by opti-

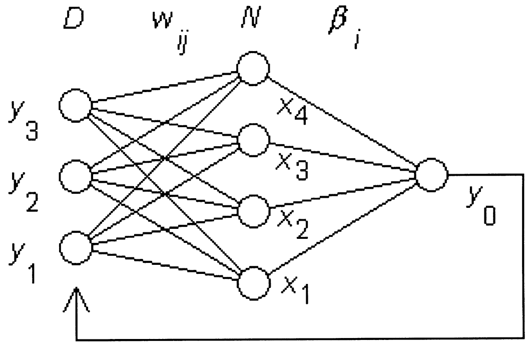
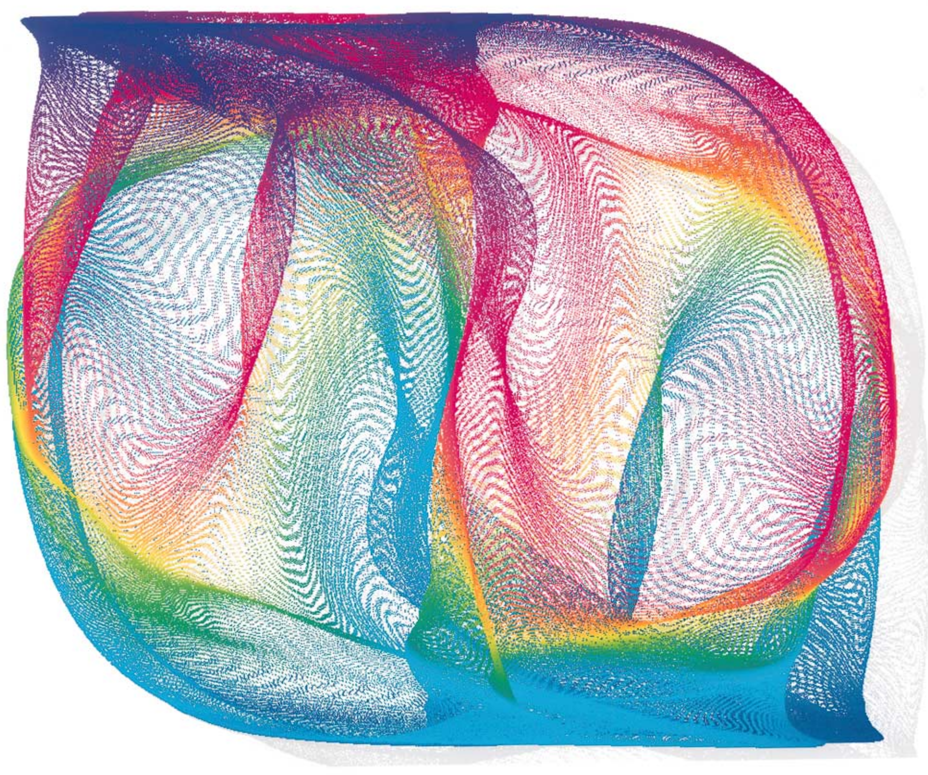
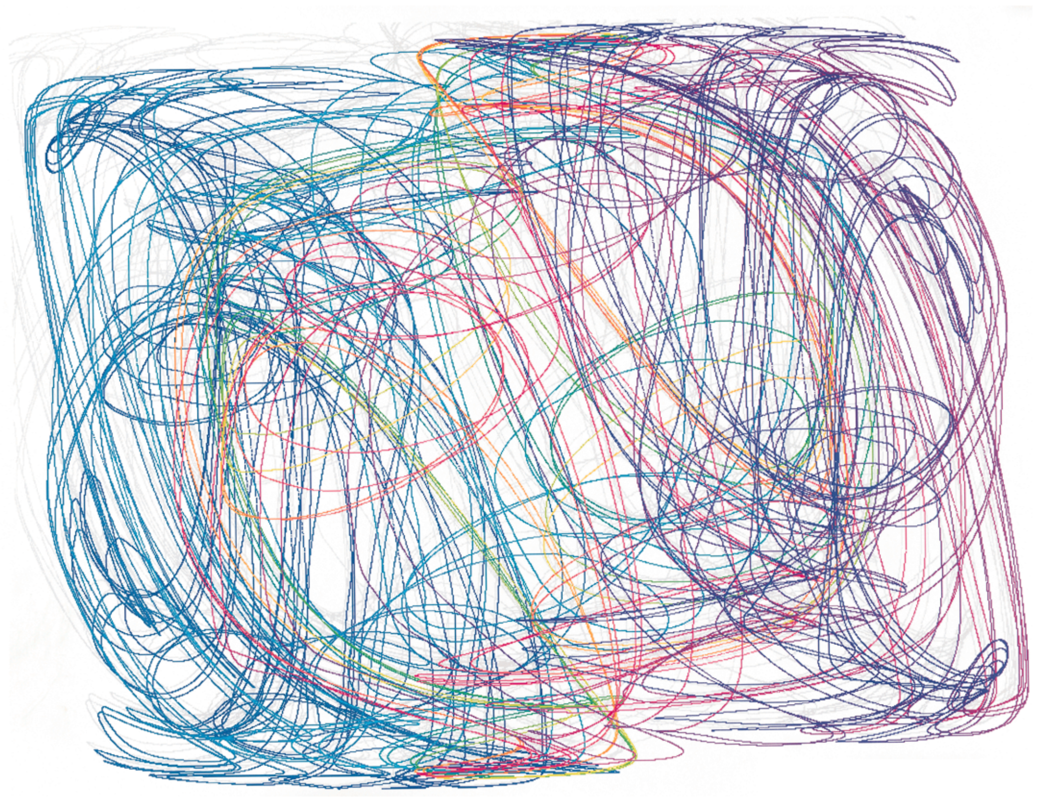


Fig. 1. A feed-forward neural network with D inputs, N neurons, and a single output that is fed back to the input.

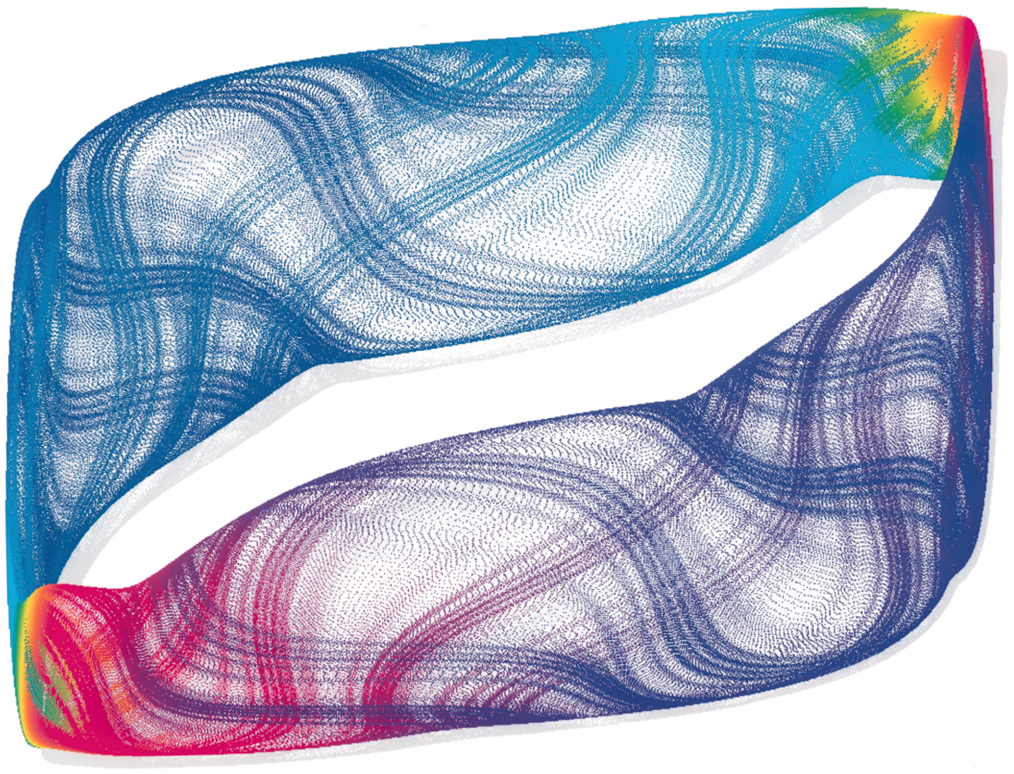
(a)



(b)



(c)



(d)

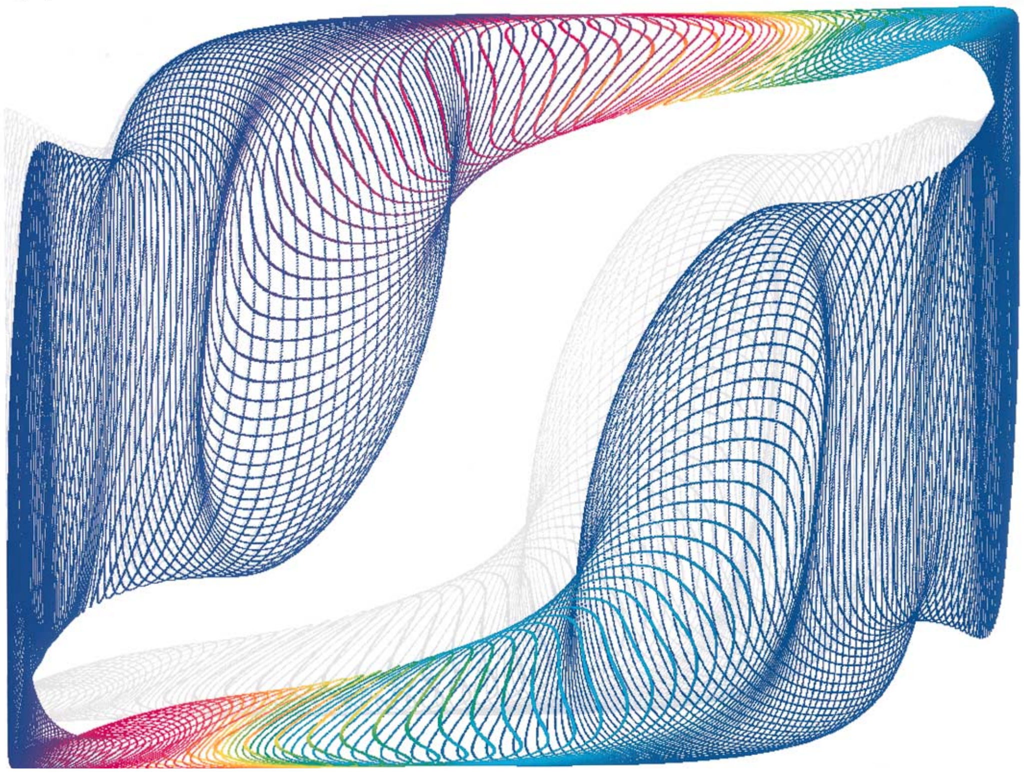


Figure 2. Sample neural network attractors.

mizing w_{ij} and βi though back-propagation [6], and the parameters D and N would have to be chosen sufficiently large to give the network adequate intelligence but not so large that computation is inordinately slow.

Rather than use the network to recognize a pattern, the goal is to produce a pattern. For this purpose it is necessary to add feedback so the network becomes a dynamical system, producing an unlimited sequence of values at its output y_0. One way to add such feedback is to let the input values represent a sequence of previous values of the function y, with y_1 being the most recent, y_2 the second most recent, and so on. With each iteration of the network, y_0 replaces y_1, y_2 replaces y_3, etc., and y_D is discarded. Thus y is a scalar dynamical variable whose value is uniquely determined by its D predecessors, where D is the time-delayed embedding dimension. However, rather than plotting successive values of y, x will be used as the dynamical variable. This has the virtue that the variables to be plotted are always in the range $(-1, 1)$ and one can independently control the number of variables to be plotted (N) and the embedding dimension (D).

For the cases shown values of w are random and uniform over the interval $(-1, 1)$ and the values of β are random and uniform over the interval $(0, 1)$. For each image the weights are chosen initially and held fixed throughout the calculation. N was chosen as 4 so that two of the values may be used as the horizontal and vertical coordinates, one of the values as the height (displayed as a shadow below and to the right), and the fourth to map linearly to a palette of colors. The palette used in the images here is just a sinusoidal variation of red, green, and blue, with phases 120 degrees apart to produce a color rainbow. Initial conditions were chosen with all y values equal to 0 and all x values equal to 0.5. The first few thousand iterates were discarded to help ensure that the orbit has converged to the attractor. The cases shown have either $D = 16$ and $s = 0.5$, or $D = 32$ and $s = 0.25$. If s is too small the network behaves linearly, resulting in an orbit that usually settles to a fixed point. If s is too large the neurons are driven into saturation and the dynamics is a cyclic sequence of points. Intermediate values give the most interesting dynamics and produce chaotic solutions with a probability that approaches 100% in the limit of large D [7], albeit with small values of their largest Lyapunov exponent. For the values chosen, cyclic points, limit cycles, tori (quasi-periodic orbits), and chaotic (strange) attractors occur with roughly equal frequency. The cyclic points and limit cycles are discarded by simply imposing a criterion on the minimum number of screen pixels that must be visited by the orbit. Typically, the calculation is terminated and restarted with new connection strengths if more than 90% of the points fall on previously visited screen pixels.

A fully operational computer program that carries out the steps outlined above is given in the Appendix.† The program should run without modification under Microsoft QBASIC, QuickBASIC, and VisualBASIC for MS-DOS. It assumes VGA (320×200-pixel, 256-color) graphics. The images in Fig. 2 were produced with an enhancement of this program that differs only in that it uses a screen resolution of 1024×768‡ and it saves each image to a graphics (PCX) file with a randomly chosen file name. Figure 3 shows a variety of other cases produced by the program.§ These images each have one million iterations and were selected from several thousand cases that were produced in several overnight runs on a 200 MHz Pentium Pro computer. Most of these cases are tori or very weakly chaotic strange attractors.

There are many ways in which this technique could be extended. The color palette could be manipulated in more imaginative ways. For example, with two additional neurons, the red, blue, and green values could be controlled by separate neurons. The 8-bit graphics could be extended to 24-bits or higher. The hyperbolic tangent could be replaced with a different function such as the anti-symmetric logistic function,

$$L(u) = 4u(1 - |u|). \tag{4}$$

Different architectures could be explored such as vector networks with a matrix of β s and D outputs, each fed back to its respective input, or networks with more than one layer of neurons, or networks with neurons not arranged in layers or connected asymmetrically.

The networks explored here are completely untrained. Thus the patterns resemble what might be produced by a child or perhaps by a monkey with a paintbrush. An interesting project would be to quantify the aesthetic quality of the patterns and train the networks to produce additional images that would be more appealing than those produced using random weights. Such a human–computer collaboration foretells the kind of interactions that we can expect to become increasingly common as computers become more powerful and people find more imaginative uses for them.

Acknowledgements—I am grateful to Dee Dechert and Dave Albers for many useful discussions of neural net dynamics and to Cliff Pickover for printing the figures.

† Also available on the World Wide Web at http://sprott.physics.wisc.edu/software.htm.

‡ High-resolution graphics drivers for QuickBASIC and other languages are available from Zephyr Software, P.O. Box 7704, Austin, Texas 78713-7704.

§ Additional images of this type can be found on the World Wide Web at http://sprott.physics.wisc.edu/fractals/neural/

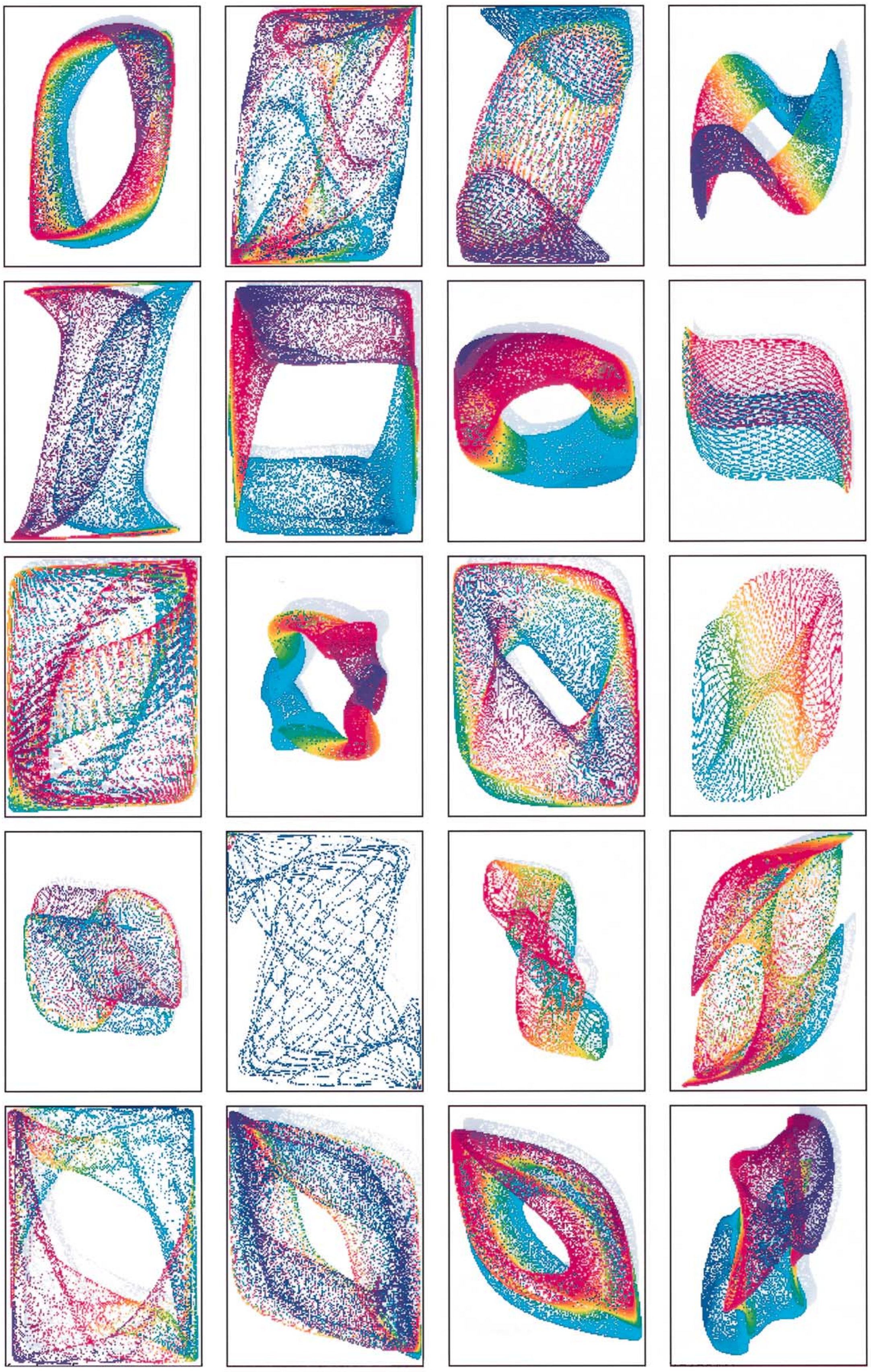


Fig. 3. Additional neural network attractors.

REFERENCES

1. J. C. Sprott, *Strange Attractors: Creating Patterns in Chaos*, M&T Books, New York, 1993.
2. J. Hertz, A. Krogh and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City, CA, 1991.
3. Hornik, K., Stinchocombe, M. and White, H., Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 1990, **3,** 535–549.
4. M. Bongard, *Pattern Recognition*, Spartan Books, New York, 1970.
5. C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.
6. D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning representations by back-propagating errors. In *Neurocomputing: Foundations of Research*, ed. J. A. Anderson and E. Rosenfeld. MIT Press, Cambridge, MA, 1986, pp. 673–674 and 696–699.
7. D. J. Albers, J. C. Sprott and W. D. Dechert, Dynamical behavior of artificial neural networks with random weights. In *Intelligent Engineering Systems Through Artificial Neural Networks*, Volume 6, ed. C. H. Dagli, M. Akay, C. L. P. Chen, B. R. Fernandez

Appendix opposite

APPENDIX

```
SCREEN 13
N% = 4                  'Number of neurons
D% = 16                 'Number of inputs (dimension)
s = .5                  'Scaling factor (network gain)
tmax& = 80000           'Number of iterations
sw% = 319               'Screen width - 1
sh% = 199               'Screen height - 1
nc% = 254               'Number of colors - 2
DIM w(N%, D%), B(N%, D%), x(N%), y(D%), PAL&(nc% + 1)
PAL&(0) = 65536 * 63 + 256 * 63 + 63    'PAL&(0) IS WHITE
PAL&(1) = 65536 * 55 + 256 * 55 + 55    'PAL&(1) IS GRAY
FOR i% = 2 TO nc% + 1
    B% = INT(32 + 32 * COS(.02464 * i%))
    G% = INT(32 + 32 * COS(.02464 * i% + 4.1888))
    R% = INT(32 + 32 * COS(.02464 * i% + 2.0944))
    PAL&(i%) = 65536 * B% + 256 * G% + R%
NEXT i%
RANDOMIZE TIMER
WHILE INKEY$ <> CHR$(27)
    CLS
    PALETTE USING PAL&(0)
    p& = 0
    FOR i% = 1 TO N%
        FOR j% = 1 TO D%
            w(i%, j%) = 1 - 2 * RND
        NEXT j%
        B(i%, 1) = s * RND
        x(i%) = .5
    NEXT i%
    FOR t& = 1 TO tmax&
        y(0) = 0
        FOR i% = 1 TO N%
            y(0) = y(0) + B(i%, 1) * x(i%)
        NEXT i%
        FOR j% = D% TO 1 STEP -1
            y(j%) = y(j% - 1)
        NEXT j%
        FOR i% = 1 TO N%
            u = 0
            FOR j% = 1 TO D%
                u = u + w(i%, j%) * y(j%)
            NEXT j%
            x(i%) = 1 - 2 / (EXP(2 * u) + 1)
        NEXT i%
        IF t& > tmax& / 50 THEN
            IF 10 * p& + 50 < t& - tmax& / 50 THEN t& = tmax&
            x% = .5 * (sw% + sw% * x(1))
            y% = .5 * (sh% - sh% * x(2))
            z% = .025 * (sw% + sw% * x(3))
            c% = 2 + INT(nc% * (.5 * x(4) + .5))
            IF POINT(x%, y%) < 2 THEN p& = p& + 1
            IF c% > POINT(x%, y%) THEN PSET (x%, y%), c%
            x% = x% + z%
            y% = y% + z%
            IF POINT(x%, y%) = 0 THEN PSET (x%, y%), 1
        END IF
    NEXT t&
WEND
END
```