A NOVEL PATTERN CLASSIFICATION SCHEME USING THE BAKER'S MAP

ALAN ROGERS, JOHN KEATING AND ROBERT SHORTEN

ABSTRACT
In a previous report, it is shown how the chaotic Baker's map can be used to implement Boolean functions. This suggests that the Baker's Map can be used as the basis for a more general pattern classification paradigm. In this note, we demonstrate that this is the case by presenting a learning algorithm for training the Baker's Map based pattern-classification system presented in [1].

1. INTRODUCTION

The properties of nonlinear and chaotic systems are being investigated by many research groups, in the hope that some engineering applications will result. Indeed, the study of chaotic dynamics for general information processing applications has proceeded in a number of directions, most notably chaos-based communications, chaos-based encryption, and memory based on chaotic maps. We have shown in recent work [1, 2] how the chaotic Baker's map can be used as a natural XOR gate, by considering the variation of the Lyapunov Dimension of the chaotic attractor against different parameter values. In this note, we complete this work by developing a learning algorithm so that the system can be trained to classify patterns.

The rest of the paper is laid out as follows: In Section 2, we summarize our previous work on this subject. In Section 3, we describe the characteristics of simulated annealing algorithms, and in Section 4, we apply the simulated annealing algorithm to our chaotic Baker's map system.

2. BACKGROUND

A. THE XOR PROBLEM

The pattern recognition problem consists of designing algorithms that automatically classify feature vectors associated with specific patterns as belonging to one of a finite number of classes. A benchmark problem in the design of pattern recognition systems is the Boolean Exclusive OR (XOR) problem. The standard XOR problem is depicted in Figure 1. Here, the diagonally opposite corner-pairs of the unit square form two classes, A and B (or NOT A). From the figure , it is clear that it is not possible to draw a single straight line which will separate the two classes. This observation is crucial in explaining the inability of a single-layer perceptron to solve this problem.

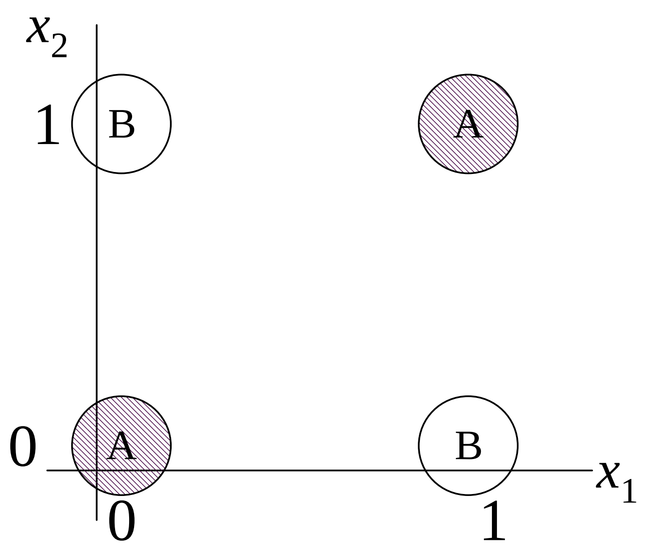

Figure 1 The Exclusive OR (XOR) Problem: Points (0,0) and (1,1) are members of class A; Points (0,1) and (1,0) are members of class B.

This problem can be solved using multi-layer perceptrons (MLPs), or by using other single-layer artificial neural networks such as the radial basis function neural network [3]. However, the inability of simple artificial neural networks, such as the Adeline [4], to solve this problem, effectively ended research interest in the area of artificial neural networks for over twenty years, which highlights the importance of the XOR problem in the design of pattern recognition systems. In this paper, we show that the Generalised Baker's Map can be trained to solve this problem in a straightforward manner.

B. The Generalised Baker's Map

In their classic study of fractal dimensions, Farmer et al.[5] introduced the Generalised Baker's Map in order to obtain rigorous results on the dimension of strange attractors. It is a transformation of the unit square [0,1]×[0,1], and has three parameters, R_1, R_2 and S:

$$x_{n+1} = \begin{cases} R_1 x_n & if\ y_n < S \\ 1/2 + R_2 x_n & if\ y_n \geq S \end{cases}$$
$$y_{n+1} = \begin{cases} y_n / S & if\ y_n < S \\ \dfrac{y_n - S}{1 - S} & if\ y_n \geq S \end{cases} \tag{1}$$

We illustrate the Baker's map transformation in Figure 2. As can be seen from (3), the mapping depends on whether the point in question is above or below a horizontal line $y = S$. All points lying in the region below $y = S$ are compressed by a factor R_1 in the x-direction and stretched by a factor $1/S$ in the y-direction. All points lying in the region above $y = S$ are compressed by a factor R_2 in the x-direction, and stretched by a factor $1/(1-S)$ in the y-direction. This entire region is then translated by $x \rightarrow x + 0.5$.

Since the Baker's Map is a mapping of the unit square, we restrict S to the range (0,1) and R_1 and R_2 to the range (0, 0.5]. In Figure 2, we show the action of the map on the entire unit square. Iterating the map gives two vertical strips, whose widths depend on R_1 and R_2. Iterating the map again gives four strips, then eight strips, and so on. The attractor is the union of a line segment (vertical direction) and a Cantor set (horizontal direction).

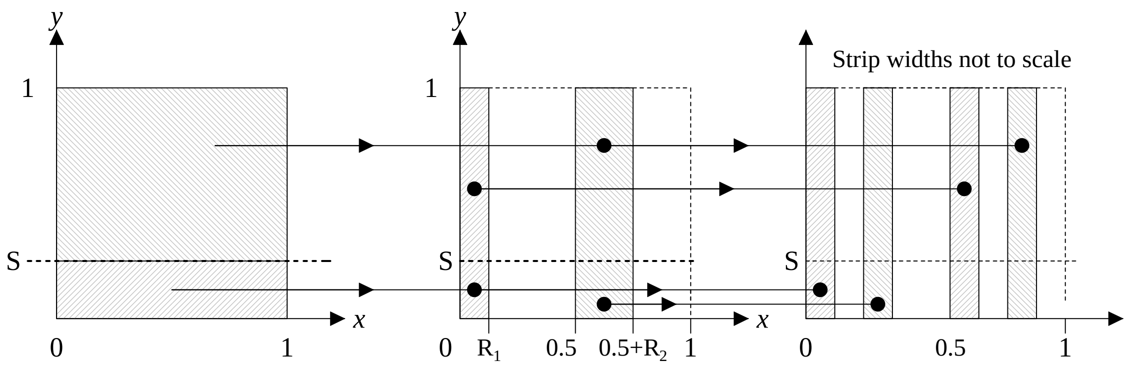

Figure 2 Action of Baker's map on unit square: Transforms square into two strips, then four strips, eight strips, and so on.

C. *Lyapunov Numbers and Lyapunov dimension of the Baker's map*

It can be seen in Figure 2 that the action of the map leads to 'stretching' in the y-direction and 'compressing' in the x-direction. It is possible to put these actions into a more mathematical framework by using the notion of Lyapunov numbers. These numbers characterise the stability of the map, and are defined as follows:

Let $J_n = [J(x_n) \cdot J(x_{n-1}) \cdot \ldots \cdot J(x_1)]$, where $J(x)$ is the Jacobian of the map, $J(x) = (\partial F / \partial x)$, for some map F.
Let $j_1(n) \geq j_2(n) \geq \ldots \geq j_p(n)$ be the magnitudes of the p eigenvalues of J_n.

Then the Lyapunov numbers are given by:

$$\lambda_i = \lim_{n \to \infty} [j_i(n)]^{1/n}, \; i = 1,2,\ldots, p \qquad (4)$$

Since the Baker's map is two-dimensional, it will have two Lyapunov numbers, characterising the average stretching/compression factors in the *x* and *y* directions (see Figure 3). Note that the *Lyapunov Exponents* are simply the logarithms of the Lyapunov numbers. It is customary to order the Lyapunov numbers, so that $\lambda_1>\lambda_2>\ldots>\lambda_n$.

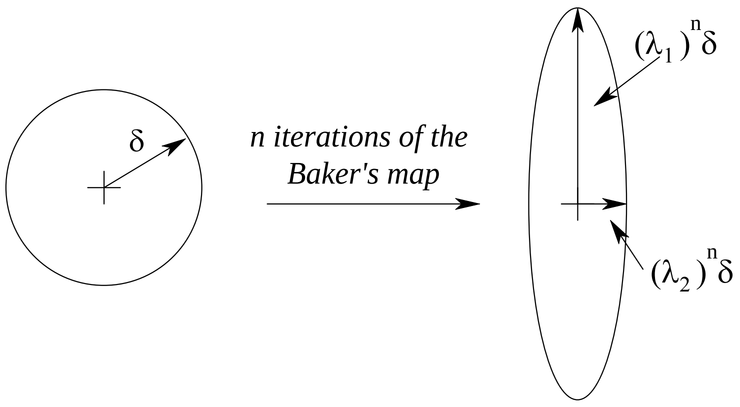

Figure 3 Lyapunov Numbers characterise the average stretching factors of some small circle of radius δ. In this case, $\lambda_1>1$ and $\lambda_2<1$.

The Lyapunov dimension was introduced by Kaplan and Yorke [6] in the so-called *Kaplan-Yorke Conjecture: that the Lyapunov dimension D_L is the same as the Information Dimension*[1] *for "typical" attractors.* For the Baker's map,

$$D_L = 1+\frac{\log\lambda_1}{\log 1/\lambda_2} \tag{5}$$

It can be shown [5] that the Lyapunov Exponents are given by:

$$\begin{aligned}\log\lambda_y &= S\log\frac{1}{S}+(1-S)\log\frac{1}{1-S}\\ \log\lambda_x &= S\log R_1+(1-S)\log R_2\end{aligned} \tag{6a, 6b}$$

In our implementation of the XOR gate, we only require two input parameters, so we shall let $R_2 = R_1$, in which case we find that:

$$\log\lambda_x = \log R_1 \tag{7}$$

From (5), the Lyapunov dimension is given by:

$$D_L = 1-\frac{\log\lambda_y}{\log\lambda_x} \tag{8}$$

[1] There are numerous ways to measure dimension (see, for example, Ott[13]. Grassberger[14], and Hentschel and Procaccia[15] defined a dimension D_q which depends on a continuous index *q*. The *Information Dimension* is the name generally given to D_1, and it takes into account the relative frequencies with which the chaotic orbit visits different regions of the attractor. (A rigorous account of D_1 is given by Ott[13].)

In Figure 4, we who how the Lyapunov (fractal) dimension) varies with R and S, and in Figure 5, we plot D_L against R, with S as a parameter. Notice that the fractal dimension varies between 1 and 2, as we would expect, and is symmetrical about S = 0.5. We have chosen slightly asymmetrical values of S in Figure 5 to illustrate this.

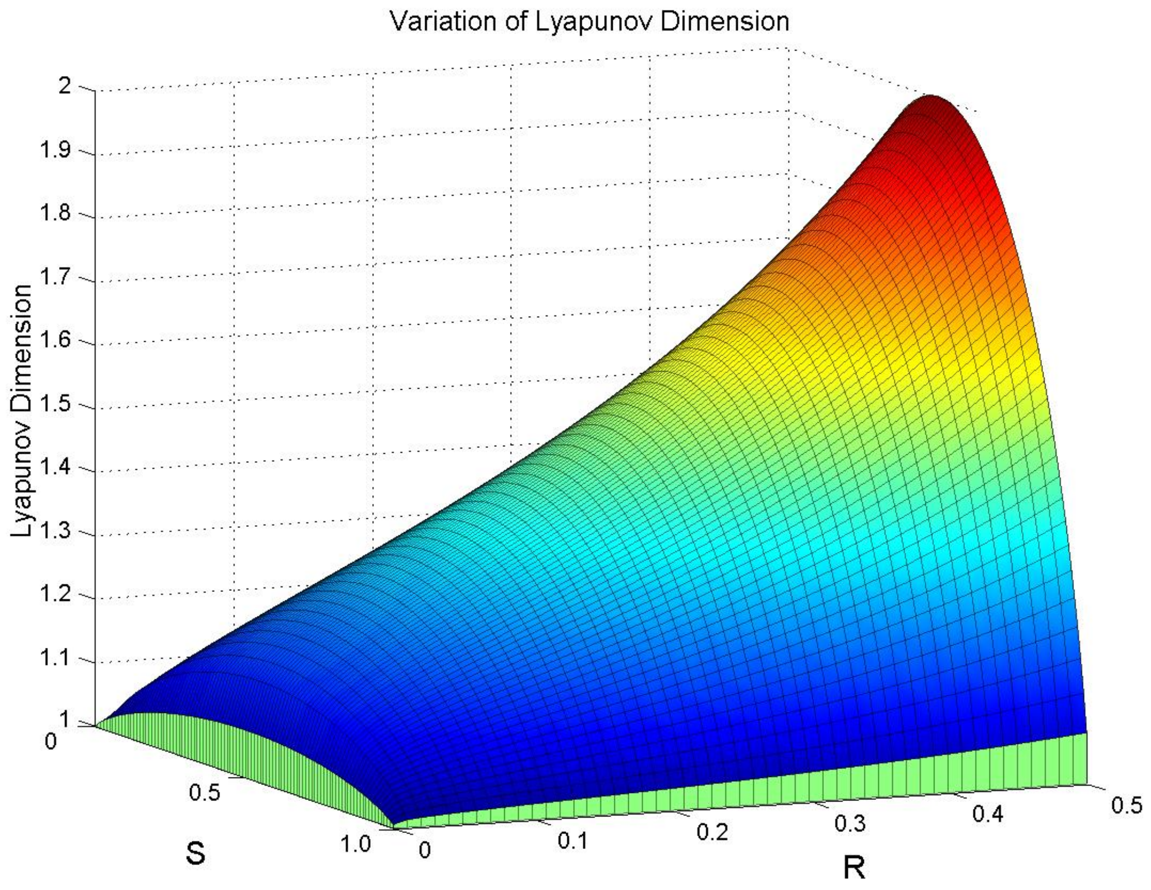

Figure 4 Variation of Fractal Dimension

We can choose values of R and S, so that a pair (low R, high S) and another pair (high R, low S) give the same fractal dimension, say D_A. This corresponds to a diagonally opposite corner pair in the XOR problem. We can say, therefore, that if the fractal dimension $D_L = D_A$, then the inputs are in class A, and if $D_L \neq D_A$, then the inputs belong to class B.

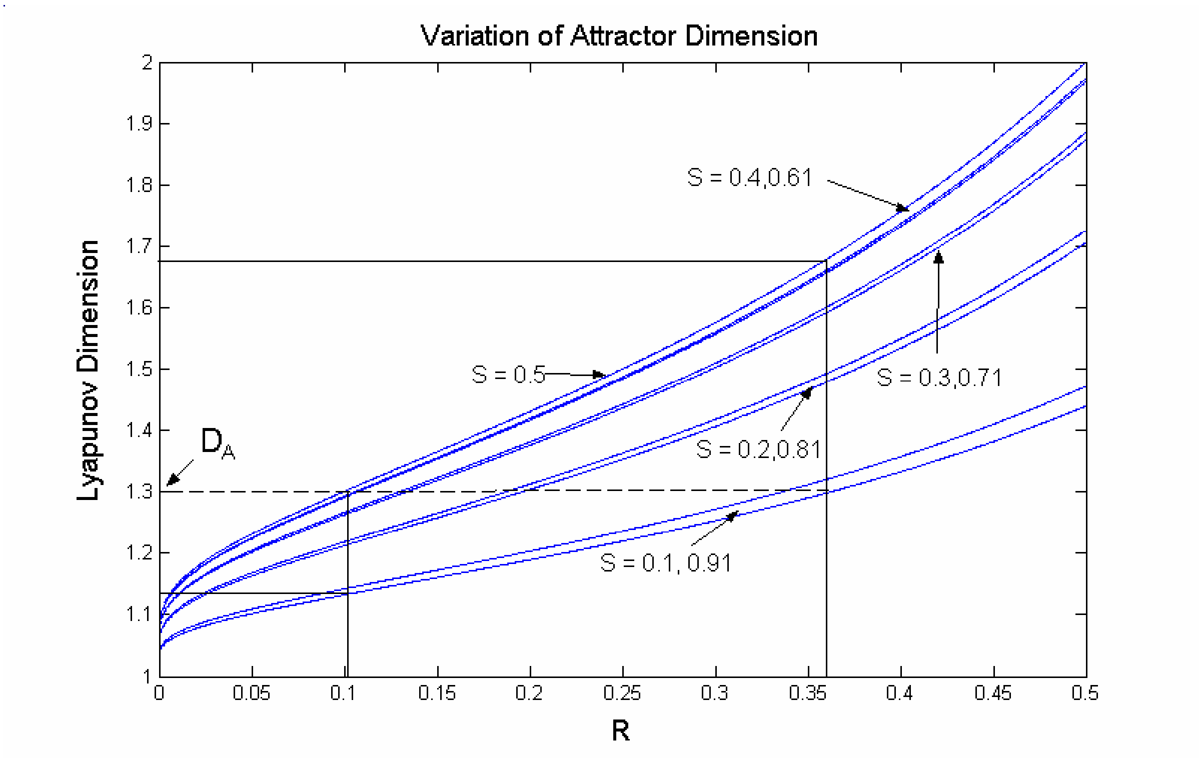

Figure 5 Variation of Fractal Dimension with varying parameter values.

Table I
Parameter values and their corresponding fractal dimension, and class, as in Figure 5.

R Value	S Value	Fractal Dimension	Class
~0.1	0.5	1.3	A
~0.36	~0.1	1.3	A
~0.1	~0.1	~1.14	B (NOT A)
~0.36	0.5	~1.68	B (NOT A)

Obviously, the points in Table I do not lie on a perfect square, but that is unimportant. The key idea is that two pairs of diagonally opposing points are mapped to the same class. It is also clear that we are quite restricted in the possible pairs of points which we can map to the same fractal dimension. However, if we choose any four (R, S) pairs of points corresponding roughly to (low, low), (low, high), (high, low) and (high, high), then by drawing a straight line through the (low, high), (high, low) points and intersecting the *y*-axis, we can effectively solve the XOR problem for a much larger set of inputs. We call the intersection of this line with the *y*-axis, D_M, the (modified) Lyapunov Dimension. This is illustrated in Figure 6.

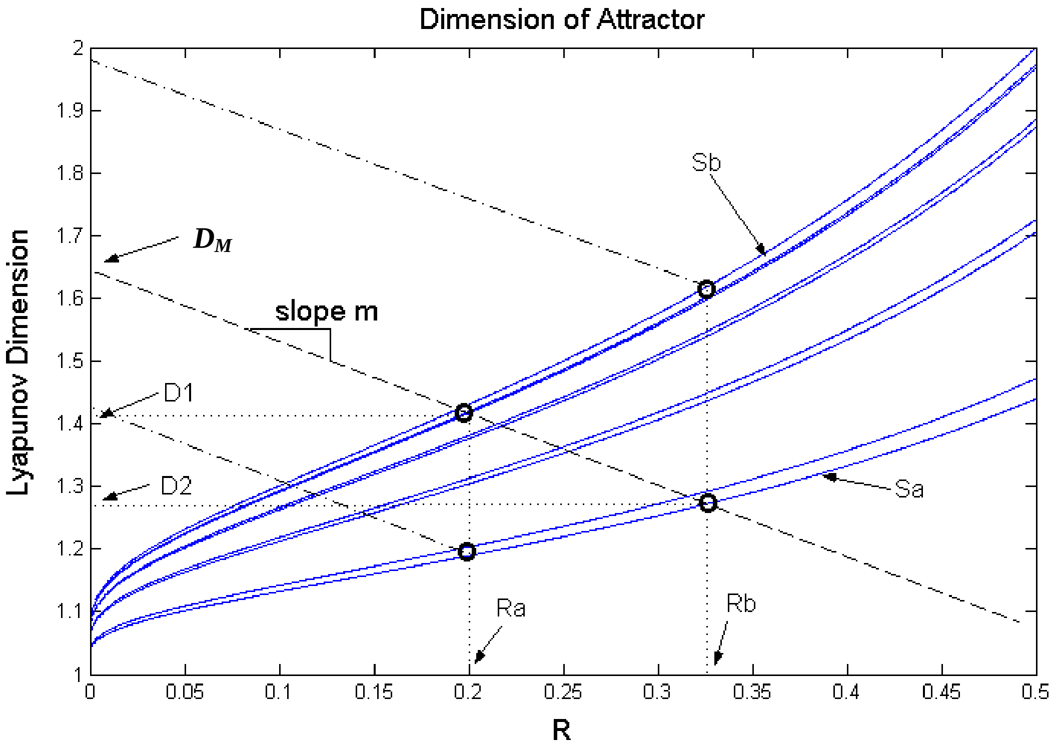

Figure 6 A more general way of solving the XOR problem: Draw a straight line through the two points belonging to class A (say), and find where the line intersects the *y*-axis.

Procedure for calculation of D_M:

(i) Given four points in the R-S plane, select the two points belonging to the same class: (R_a, S_b), (R_b, S_a) in the Figure drawn above.

(ii) Calculate the Lyapunov dimensions corresponding to the two points, called D_1, D_2.

(iii) Calculate the slope, $m = (D_1 - D_2)/(R_a - R_b)$.

(iv) The dimension $D_M = D_1 + m.R_a = D_2 + m.R_b$.

As D_M is constantly calculated, we can tell whether the inputs are in class A, or not. An algorithm of this form is referred to as a training algorithm in the artificial neural network and statistical pattern recognition literature [7]. The availability of such an algorithm, and its complexity, ultimately determines the applicability of a particular paradigm for a given problem. In our case, given a set of class labels, and a set of vectors, the training parts of the pattern recognition problem is trivial, involving only the simple calculation of a slope. For an ANN, solving this problem requires repeated calculation of the slope for at least two hyperplanes, and so is more computationally intensive.

In order to generalise the system, we would like an algorithm which, if given the R-S points and the class labels, would calculate the slope m and intercept D_M

automatically. This task can be formulated as an optimisation problem, and can be accomplished using several different methods. For illustration purposes, we shall use a simulated annealing algorithm, and we describe the algorithm in the next section. (There are, of course, many other possible algorithms which would achieve the same result.)

3. SIMULATED ANNEALING

A. Methodology

Annealing is a process used to toughen steel, so that it may be machined or cold-worked [8]. It involves heating a solid to a high temperature, and then slowly cooling it in a controlled manner. At high temperatures, the molecules have a lot of energy and are able to move randomly within the solid. They tend to move to positions that lower the energy of the system as a whole, but can also move to positions of higher energy, with a probability $e^{-\Delta E/T}$, where ΔE is the change in energy of the system, and T is the temperature of the system. The cooling process wipes out any traces of previous structure, and relieves internal stresses within the metal, making it less likely to fracture. With an absence of defects, the metal crystal is in a global minimum energy state.

Metropolis et al.[9] proposed a simulation scheme for the evolution of thermodynamic systems to equilibrium, in 1953. Thirty years later, Kirkpatrick et al.[10] realized that the Metropolis algorithm could be applied to optimisation problems in general, with a cost function taking the place of energy. Simulated annealing is particularly suited to optimisation problems where the global minimum is located amongst many poor local minima [11]. In order to apply the Metropolis Algorithm, we require the following elements [12]:

- (i) A description of possible system configurations
- (ii) A method of randomly perturbing the system configurations
- (iii) A cost function (analog of energy), whose minimisation is the aim of the procedure
- (iv) A control parameter T (analog of temperature) which determines the likelihood of an increase in cost being accepted.

The simulated annealing algorithm then follows these basic steps:

1. Initialise the system with some state S, and control parameter $T = T_0$.
2. Perturb system randomly to a new state S_N.
3. Determine change in cost function, $\Delta E = E(S_N) - E(S)$, due to the random perturbation.
4. If $\Delta E < 0$, accept new system state S_N, OR, if $\Delta E > 0$, accept S_N with probability $e^{-\Delta E/T}$.

5. Repeat steps 2 to 4 for, say, 100 successful reconfigurations.
6. Let T = T * c, where c < 1, and then repeat steps 2 to 5.
7. Stop when T gets small, or E cannot be reduced further.

The annealing schedule, which determines how many perturbations are carried out at a given temperature, and the value of the control parameter c, is usually determined through experimentation. The initial temperature T_0 is chosen so that $\exp(-\Delta E/T_0) \approx 1$, that is to say, most random perturbations leading to an increase in energy will be accepted. Herein lies the power of simulated annealing – the system is able to escape from local minima. As the temperature is lowered, only small perturbations are accepted, and so the system gradually approaches equilibrium.

In our system, we wish to position a line so that it will either map two patterns to the same modified Lyapunov dimension D_M, or separate patterns so that we can say that values of D_M less than a certain value belong to, say, Class A, and all others are class B.

For illustration purposes, we show three possible situations:

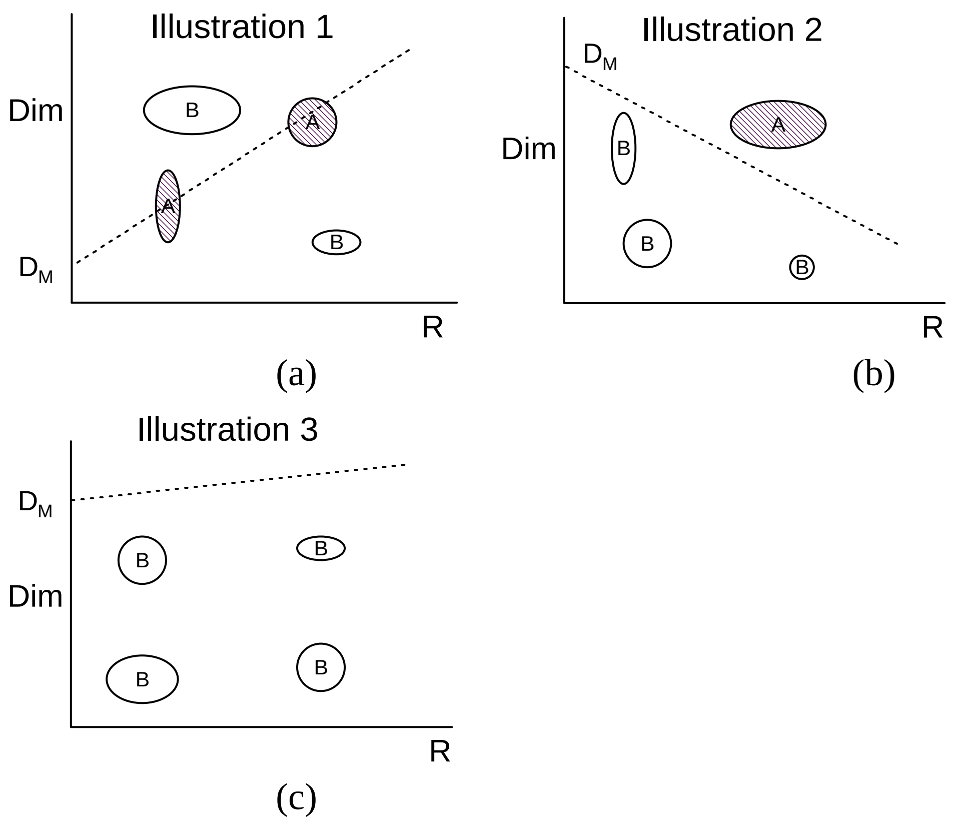

Figure 7: Three possible class configurations (out of a possible 16)

The three illustrations show different class configurations in the R- D_L plane (with S as a parameter). Illustration 1 is simply an XOR gate. Illustration 2 is an AND gate. Illustration 3 is an inverting logic gate (every value of D_M leads to a NOT Class A outcome). Obviously, there are 16 possible variations on this theme. We seek a way of training the dotted line in the figures so that it correctly classifies the patterns presented to it.

If we consider a line given by $ax + by + c = 0$, the perpendicular distance from a point x_1,y_1 to the line is given by:

$$d = \frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}} \tag{9}$$

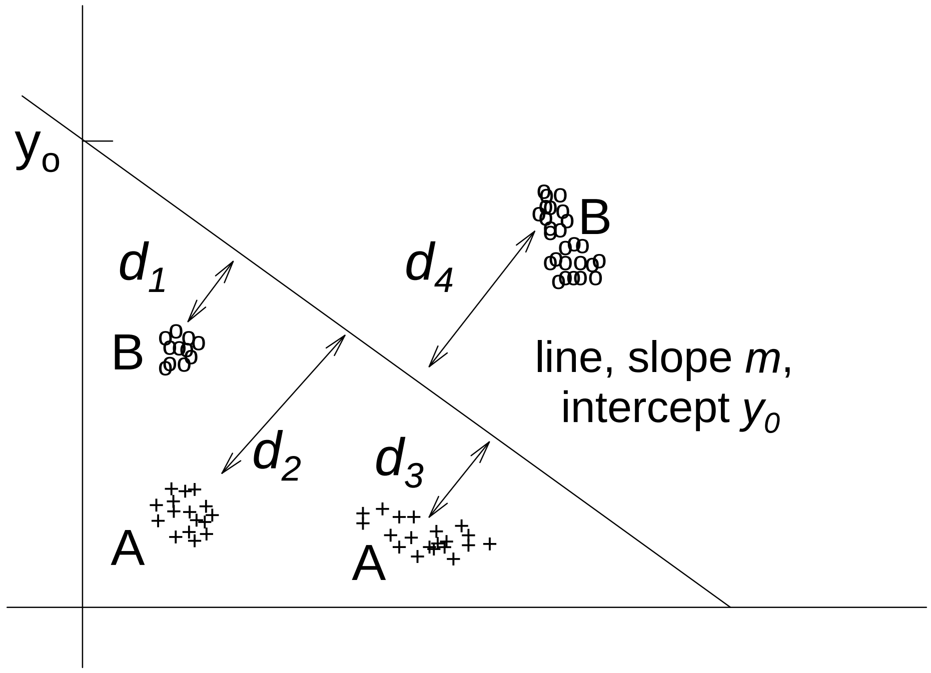

Figure 8: Distances from each pattern cluster to the separating line. '+' symbols are patterns belonging to Class A; 'o' symbols belong to Class B.

B. Training and Testing of the Classifying Lines

In the pattern classification literature, it is usual to train the classification scheme using a certain set of patterns, and then test the system with a different set of patterns to see if these are classified correctly. This shows that the system can generalise — that it can classify patterns that it hasn't seen before. In our scheme, we will train two different classifying lines to take account of the possibility of an XOR-type arrangement of the pattern classes. In the testing process, we choose the classifying line that gives the fewer classification errors.

Consider a set of patterns and classes as in Figure 8 above, with a line, slope *m*, and y-intercept y_0 lying in the plane. Considering each pattern in turn, if the point lies on the correct side of the line for classification purposes, we say that the distance is negative, otherwise it is positive. In Figure 8, the line classifies patterns in three of the clusters correctly. Hence we have d_2, d_3, $d_4 < 0$, and $d_1 > 0$. We use a cost function based upon the hyperbolic tan function (see Figure 10), where we sum over all the patterns in all the classes.

Cost Function 1: $E = \sum_{class\,A,B} \tanh(d_i)$

If the pattern classes are in an XOR-type arrangement, we just consider one of the pattern classes, and using simulated annealing to train a line which runs through the centroid of both pattern clusters. The cost function we use has the following form:

Cost Function 2: $E = \sum_{class\,A} \tanh(d_i)^2$

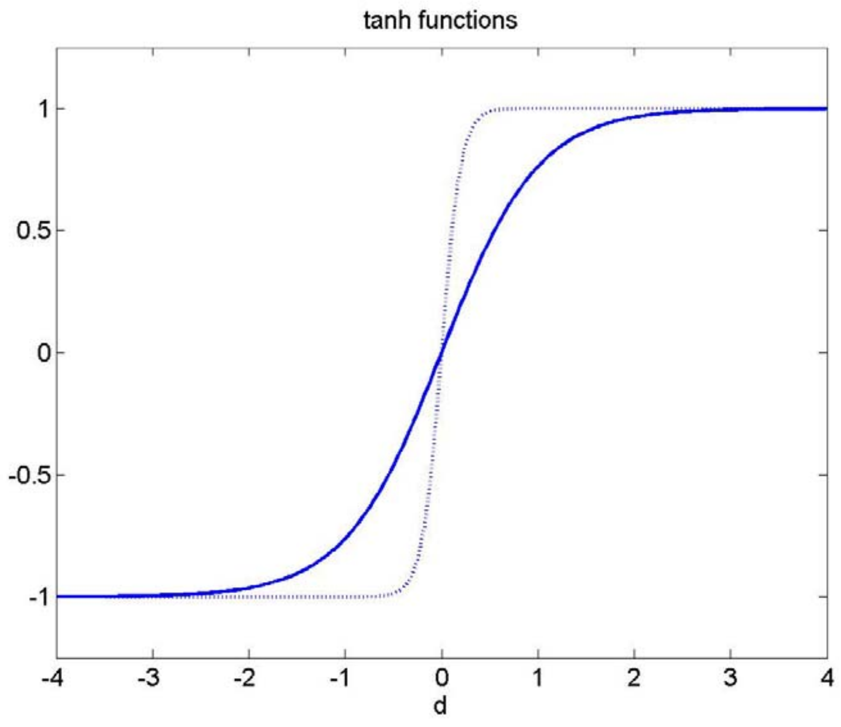

Figure 10   Tanh functions: Solid line is tanh(d) ; dotted line is tanh(5d)

When the annealing algorithms for both cost functions have converged to a solution (a slope and an intercept), we test the two solutions using a new set of patterns. The correct classifying line should give a zero or minimal error.

As an illustration of the simulated annealing process, we will demonstrate with a set of classes as in Figure 11. The desired line should separate the two classes as shown. We will choose an initial line with slope $m = 2$ and y-intercept = -3. We determine the perpendicular distances from points scattered randomly about the centroids of the individual patterns (0,0),(1,0),(0,1) and (1,1), and determine if the line is orientated to the correct side of each of the patterns.

In Figure 12, we show an energy landscape of the system, where the elevation represents the cost of a particular line. The uplands represent regions where all patterns are classified incorrectly, and the basin represents the desired line, which in this case has slope of –1 and intercepts the y-axis at 1.55. The simulated annealing algorithm will find the minimum energy point of this landscape.

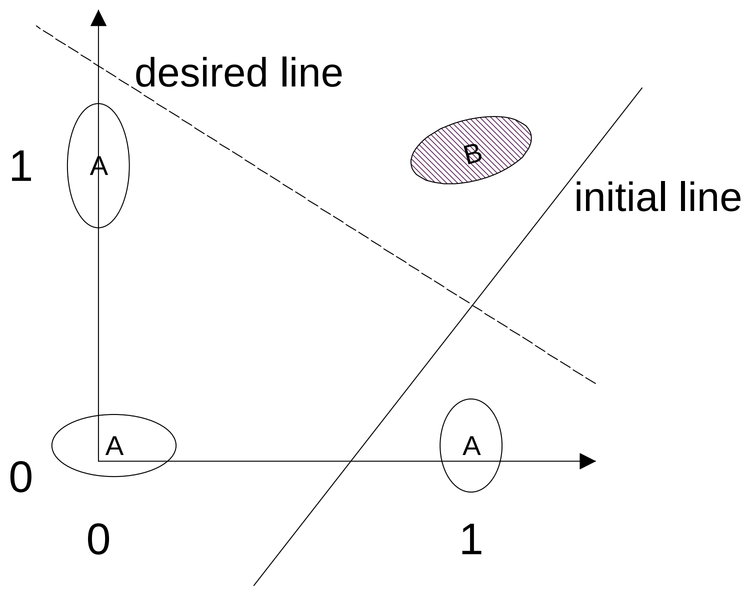

Figure 11: Illustrative example of Simulated Annealing. The Initial line has slope $m = 2$, and y-intercept = -3. The desired line should have slope $m = -1$, and y-intercept > 1

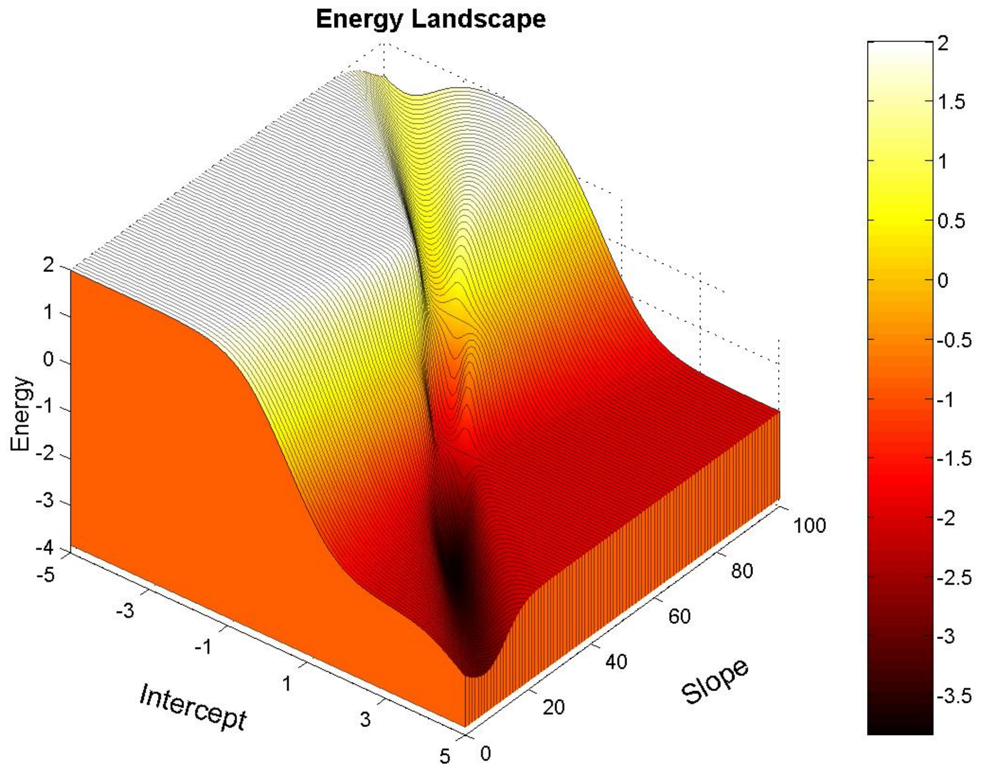

Figure 12: Energy Landscape for patterns in Figure 11. Minimum Energy = -3.84, when m = -1 and y_0 = 1.55.

Note that for simplicity in this example, we only used four patterns, and the minimum value of cost function is, as a consequence, approximately –4. For cases where we have several hundred patterns, the minimum energy would be considerably lower.

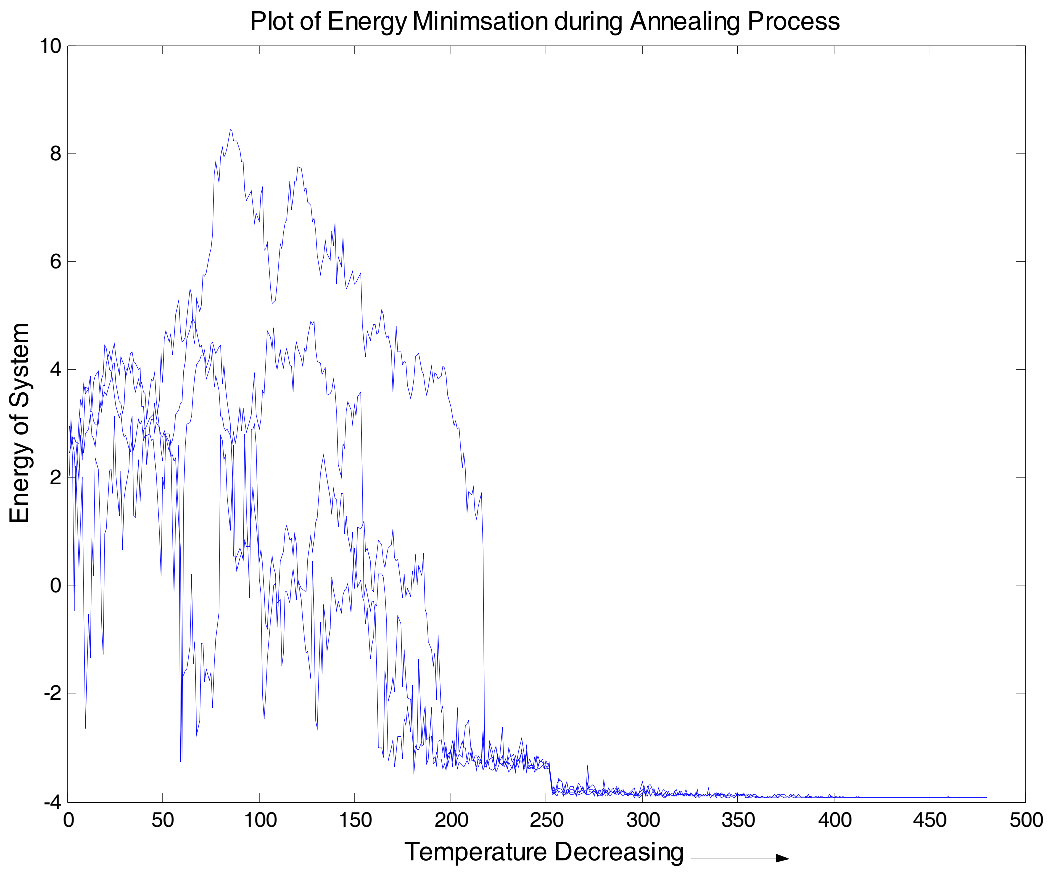

Figure 13: Plot of Energy (Cost) as Temperature is decreased, using simulated annealing algorithm.

In Figure 13, we show four different plots of energy during the annealing process. Clearly, there is quite a lot of variation because it is a random process, however the final value of energy is very close to –4 in each case. It is advantageous to introduce an extra condition into the cost function to ensure that the values of slope and intercept remain small:

$$E = \sum_{i=1}^{4} \tanh(d) + |m| + |y_0|.$$

In Figure 13, this condition is relaxed near the "250" mark on the x-axis, and accounts for the small drop in energy at that point. The choice of cost function depends on the type of patterns we wish to classify, as there may be occasions where we need a large slope.

A typical final output from the algorithm is:

System Energy = -3.93811
Slope = -0.970605
Y-Intercept = 1.50911

4. TRAINING OF CHAOTIC BAKER'S MAP SYSTEM

In [1], we showed how the system could be used as an XOR gate, and an AND gate, and we also combined these into a half-adder. In fact, there are 16 possible 2-bit patterns which we could classify using our 2-parameter Baker's Map System, as shown in Figure 14. Each value of R and S can be high or low, and so defining Class A or B to be either high or low, we essentially have all 16 possible 2-bit logic gates.

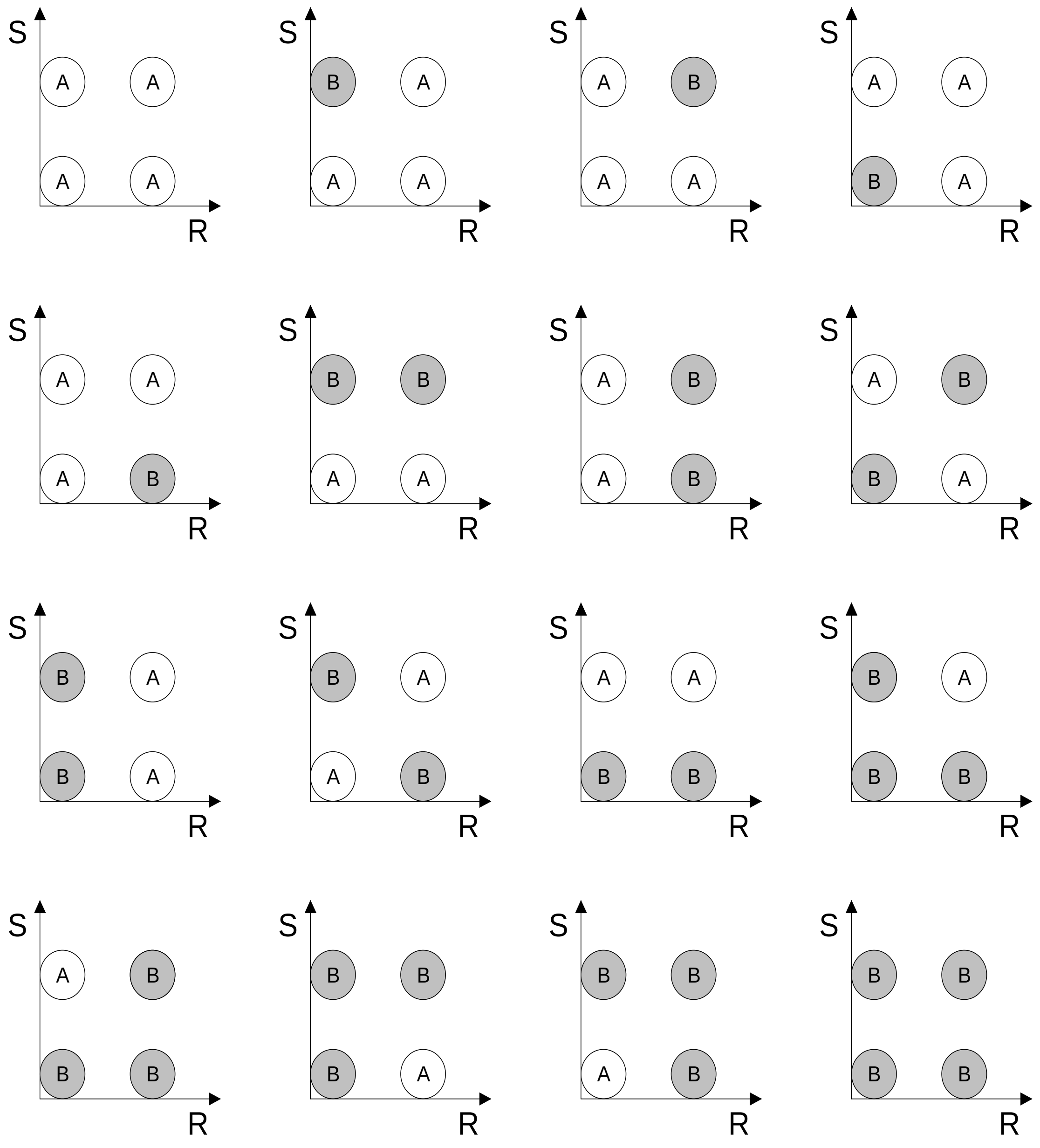

Figure 14 All possible 2-bit Pattern Classes

We will illustrate the training procedure for two representative patterns, though by suitable choice of cost function, we can train the slope to classify any of the patterns in Figure 14.

A. Example 1

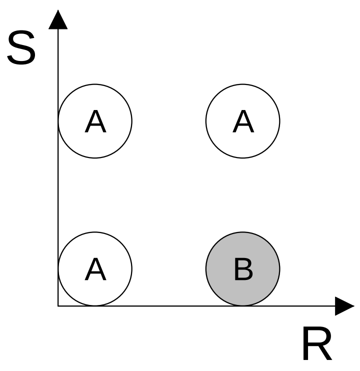

Figure 15 Pattern Classes for example1.

It is necessary to assign values to R and S, representing high and low. We arbitrarily choose the following values (though limiting both R and S to (0,0.5).

Pattern	R Value	S Value
0 0	0.1	0.1
0 1	0.1	0.45
1 0	0.45	0.1
1 1	0.45	0.45

The training procedure is carried out on a line in the R-D_L plane, so we find the corresponding values of Lyapunov Dimension D_L for each (R,S) pair. We can do this because there exists a closed-form expression for D_L in terms of R and S [5].

$$D_L = 1 - \frac{S \ln[1/S] + (1-S)\ln[1/(1-S)]}{\ln R}$$

R-Value	S-Value	D_L
0.1	0.1	1.141182
0.1	0.45	1.2988549
0.45	0.1	1.4071
0.45	0.45	1.8618

We now apply the simulated annealing algorithm to find a line which will classify the patterns correctly, as in Figure 16. Then we apply test patterns to the Baker's map, to verify that the correct patterns are detected.

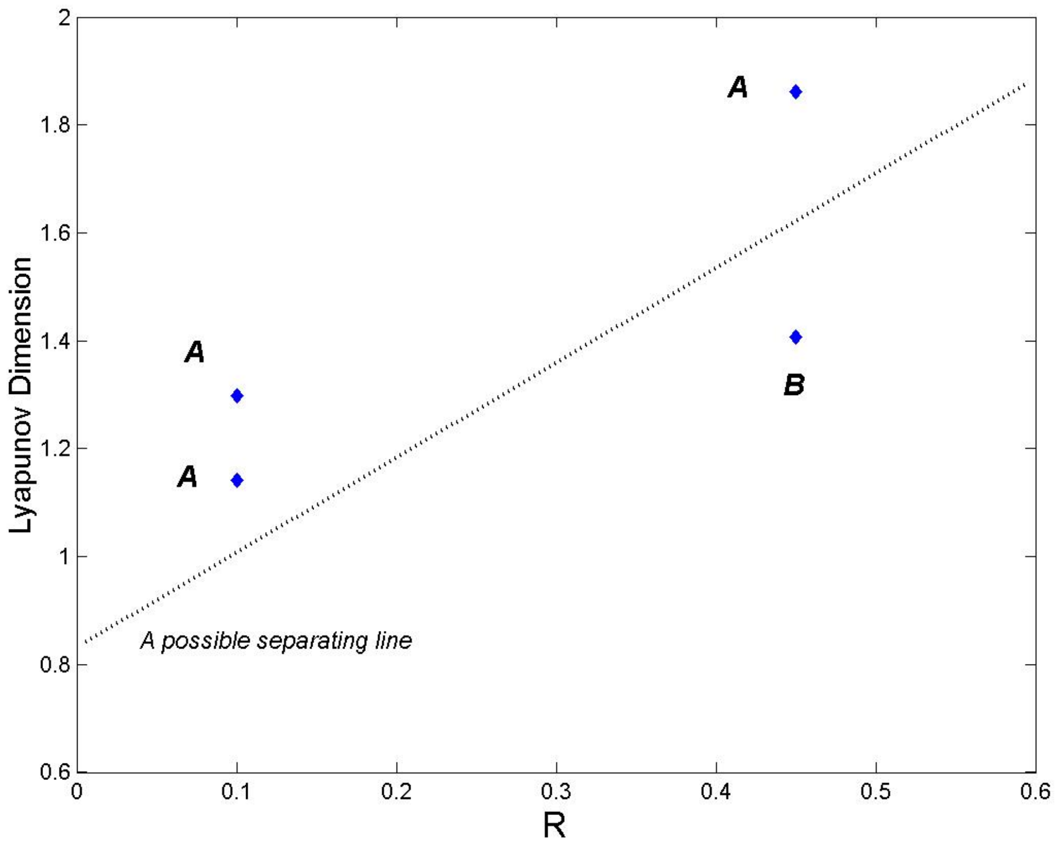

Figure 16 Pattern Classes mapped into the R-D_L plane.

When we apply the simulated annealing algorithm, we find that the slope *m* = 2.14831 and *y*-intercept = 0.620018. This information is used to separate the classes, as shown in Figure 16. The Lyapunov numbers, (and hence the Lyapunov Dimension) are jumpy because of the nature of the map (eqn 1.above). We find average values of the Lyapunov Dimension, using a 500-point averaging window. In Figure 17, we apply the four possible patterns to the system, as inputs. After modifying the Lyapunov dimension, points lying below the separating line correspond to the pattern Class B.

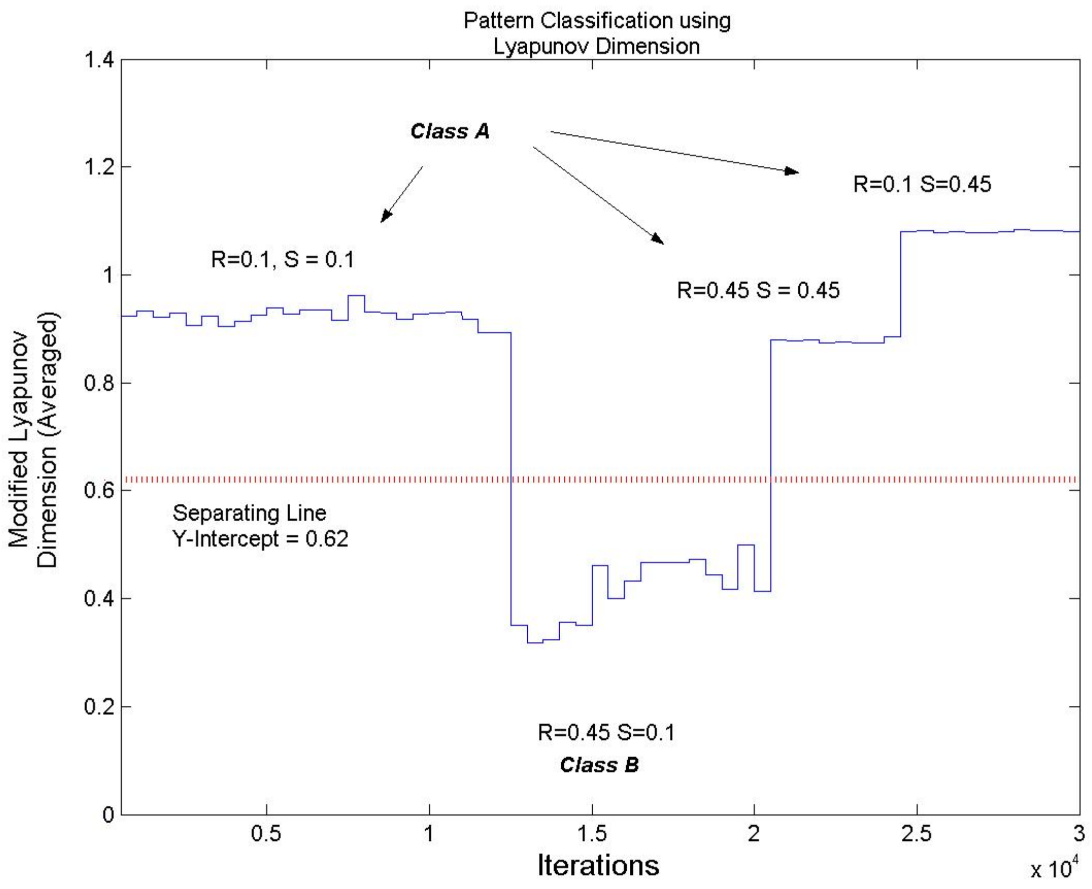

Figure 17 Patterns above the dotted line are Class A. We can classify the patterns simply by looking at their Modified Lyapunov Dimension.

B. Example 2

In the second example, we train the system to classify an XOR-type set of patterns, as shown in Figure 18.

For the cost function, we shall associate tanh functions with class B, and inverted Gaussian functions with class A, in order to give an appropriate energy minimum for the line that will classify the patterns correctly.

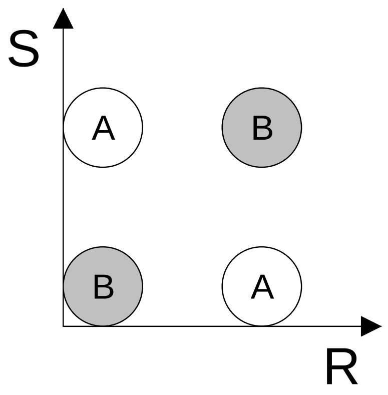

Figure 18 Pattern Classes for Example 2

Pattern	R Value	S Value
0 0	0.2	0.2
0 1	0.2	0.4
1 0	0.4	0.2
1 1	0.4	0.4

R-Value	S-Value	D_L
0.2	0.2	1.3109
0.2	0.4	1.4182
0.4	0.2	1.5461
0.4	0.4	1.7345

The simulated annealing algorithm returns the following values for slope and intercept:

Slope = 0.487
Intercept = 1.34

The modified Lyapunov Dimension will be approximately 1.34 for the patterns in Class A. In practice, the dimension will lie in a small band about 1.34. Thus by deciding if the modified Lyapunov Dimension lies within or outside this band, we can determine if the applied pattern belongs to class A or class B.

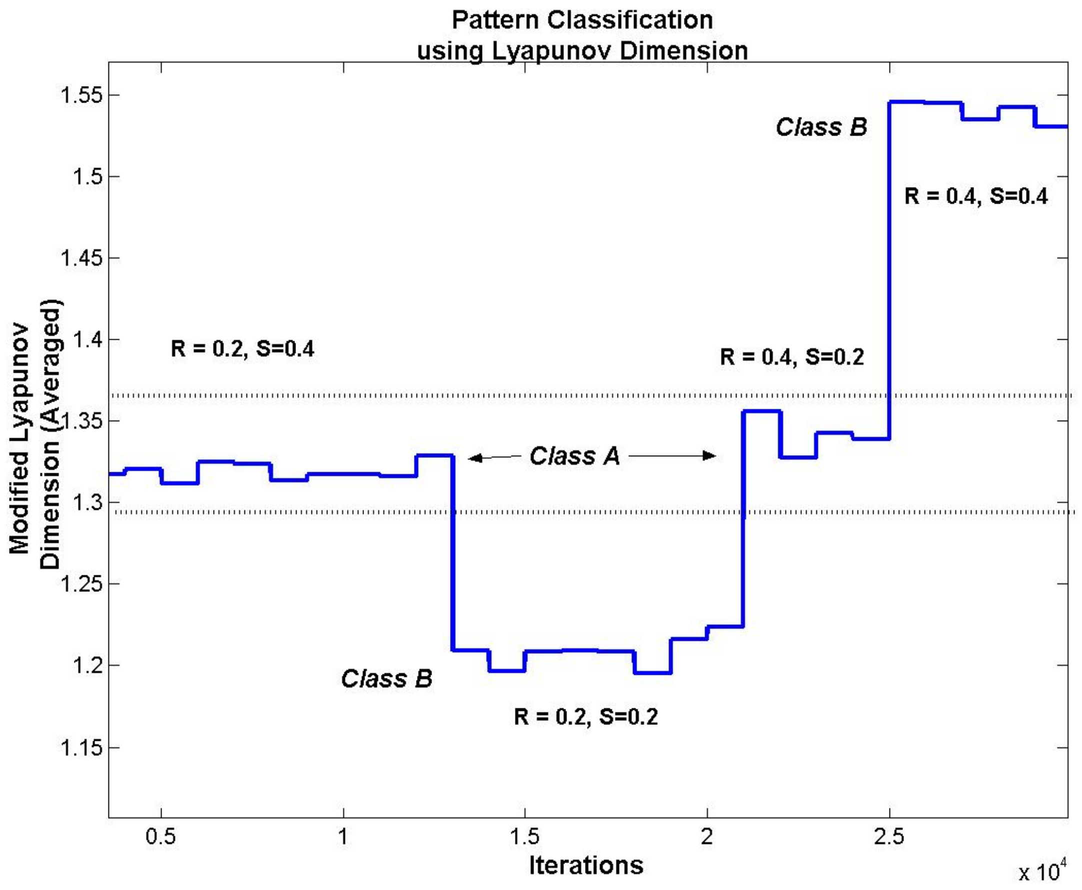

Figure 19 Pattern Classification for example 2. If the Modified Lyapunov Dimension lies within the dotted lines, the pattern is classified as Class A.

5. Conclusions

We have shown that the chaotic Baker's Map system can be trained to classify any 2-bit pattern, thus illustrating the flexibility of the approach. A simulated annealing algorithm was used to train the system, although it would be possible to use other methods, such as a perceptron algorithm, to achieve the same goal.

References:

1. A.Rogers et al. Chaotic Maps and Pattern Recognition –The XOR Problem, *Chaos, Solitons and Fractals,* 14, pp.57-70 (2002).
2. A.Rogers et al. Chaotic Maps and Pattern Recognition –The XOR Problem, *Technical Report NUIM SS0109*, NUI Maynooth, 2001.
3. C.M.Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford (1995).
4. S.Haykin. *Neural Networks—A Comprehensive Foundation*. Prentice-Hall, New Jersey (1999).
5. J.D.Farmer, E.Ott, and J.A.Yorke. The dimension of chaotic attractors, *Physica D*, 7, pp.153-180 (1983).
6. B.D.Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge (1996).
7. J.Kaplan and J.Yorke. Chaotic behaviour of multidimensional difference equations. In *Functional Differential Equations and the Approximations of Fixed Points, Lecture Notes in Mathematics* Vol.730, edited by H.O.Peitgen and H.O.Walther, pp.204-207. Springer, Berlin (1979).
8. B.H.Amstead et al. *Manufacturing Processes.* Wiley (1987).
9. N.Metropolis et al. Equation of State Calculations by Fast Computing Machines, *Journal of Chemical Physics*, 21, pp.1087-1092 (1953).
10. S.Kirkpatrick, C.D.Gelatt Jr., and M.P.Vecchi, Optimization by Simulated Annealing, *Science* 220, pp.671-680 (1983).
11. W.H.Press et al. *Numerical Recipes in C++*. Camobridge University Press, Cambridge (2002).
12. J.G.Keating and D. Noonan, The structure and performance of trained Boolean networks. In *Neural Computing: Research and Applications II*, edited by G. Orchard. Irish Neural Networks Association, Belfast, pp. 79-86 (1994).
13. E.Ott, *Chaos in Dynamical Systems.* Cambridge University Press, Cambridge (1993).
14. P.Grassberger, Generalized Dimensions of Strange Attractors, *Physics Letters A*, 97, pp.227 (1983).
15. H. Hentschel and I.Procaccia, The Infinite Number of Generalized Dimensions of Fractals and Strange Attractors, *Physica D,* 8, pp.435 (1983).