Undergraduate Projects in the Application of Artificial Intelligence to Chemistry. II Self-Organizing Maps

Hugh Cartwright

Physical and Theoretical Chemistry Laboratory, Oxford University, South Parks Road, Oxford OX1 3QZ, England, hugh.cartwright@chem.ox.ac.uk

Abstract: It is often necessary in science to identify samples that have features in common. For example, one might wish to find those NMR spectra in a large database that have similar patterns of resonances or identify samples amongst a large number of specimens of river water that analysis shows have similar biochemical oxygen demand, heavy metals concentration, organochlorine content, and so on.

The determination of relationships among samples is a task to which Artificial Intelligence is increasingly being applied. In this paper, we investigate the Self-Organizing Map (SOM), whose role is to perform just this kind of task; in other words, to cluster data samples so as to reveal the relationships that exist among them. The self-organizing map is a method, which, unusually, combines a mathematical foundation with an intuitive interpretation.

We will describe how a simple SOM operates, what kinds of data may be analyzed using one, and how a computer program to run a SOM can be written by anyone-whether student or teacher-with modest programming skills. Portions of sample source code are included in this paper, and program listings for the examples that are discussed are available in the supporting materials. The supporting files can also be used to see the maps in operation.

Introduction

Let us begin at the end, so to speak, by considering a completed map that illustrates what the SOM is aiming to produce. Figure 1 shows the result of an analysis of a typical set of data (individual color versions of all figures are available in the supporting materials). Running one of the programs described in this paper can generate this type of map.

The two-dimensional map shown in Figure 1 resembles a contour view of a land map, and it is divided into different regions distinguished by their color. Every sample within the database used to create the map can be allocated to a particular square in the map, and similar samples are allocated to squares that are close together. Similar samples are thus clustered, and this clustering of samples helps the user to identify the factors that samples have in common. In this way the map brings about a kind of rationalization of the data. The mechanism by which this is accomplished forms the focus of this paper.

Large collections of data in which there are multiple non-linear correlations between the different samples are generated routinely in chemistry; indeed, gathering and analyzing the data that constitute such sets might be regarded as one of the central activities in science.

To make full use of the data, one wants to tease out the relationships that link different samples, but extracting these relationships may be difficult, especially if each sample is made up of many separate items of data (such as the intensity of light absorption at a number of different wavelengths, or the results of elemental analysis of mixed minerals). It is not always possible to express the relationships in a readily digestible algebraic form, so numerical methods, such as Principal Components Analysis [1] may be used to seek regularities in the data. However, these methods are limited when analyzing complex data sets in that they generally yield either an abstract representation of the data or one that is cast in three, four, or more dimensions. This introduces difficulties in interpretation for any users who are not specialists in the field of numerical analysis.

Often a visual rather than an analytical representation of the data will effectively reveal how samples are related to each other, and in such cases a SOM may be used to arrange the data so that useful relationships among the parameters are presented in a readily interpreted form. A SOM is normally constructed in two dimensions, and this makes its visual interpretation particularly straightforward.

Principles of the Self-Ordering Map

The development of the self-organizing map (or self-ordering map) can be traced back to pioneering work by Kohonen [2]; in fact, such maps are the end result of analysis by what are known as Kohonen networks. The SOM is a type of neural network (to be discussed in a later paper), in which a number of identical simple processing units are linked together and function co-operatively.

The principle steps in the construction of a map are outlined in this section. Later, each step will be discussed in greater detail and illustrated with segments of Java source code. Those unfamiliar with Java should find the code relatively straightforward to understand and should have little difficulty in translating it into Fortran, Basic or other computer languages.

A SOM is a computer program whose role is to compress multidimensional data on to a map of lesser dimension (usually two). At its heart is a small memory with a simple structure. In this memory the SOM stores its accumulated knowledge at a set of *nodes*. Approximately 30 to 5000 nodes,

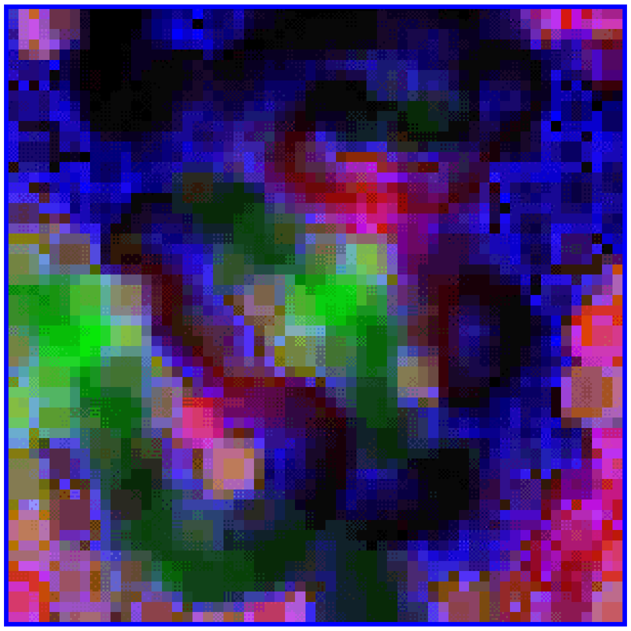

Figure 1. A typical completed self-organized map.

depending upon the scale of the problem, are arranged in a regular two-dimensional structure, in which each node has three, four or six, neighbors (Figures 2a, 2b, 2c). The nodes form a triangular, rectangular, or hexagonal array [3].

At each node is stored a set of *weights* (Figure 3). It is these weights that constitute the memory of the map; the calculations that are performed by the algorithm consist of the determination of suitable values for each weight. Once appropriate values for these weights have been found, the completed map, which typically takes the sort of form shown in Figure 1, is ready for use.

At the start of the calculation, the SOM knows nothing about the data it is to interpret, and, because the weights represent the accumulated knowledge of the map, they initially are set to random values. As the map develops, the weights gradually change according to the procedure given below. The effect is that the memory of the map, stored in these weights, slowly evolves to embody a useful representation of the data.

As the map learns about the data, it organizes it in ways that, one hopes, will reveal valuable correlations. The map receives no feedback from the user or the database; this is a type of *unsupervised learning*-in other words, the algorithm learns about the data merely by inspecting it; no outside intervention or guidance from the user is necessary.

Before considering the individual steps involved in constructing a map, the reader may wish at this stage to go to the supporting materials and run the “colourmap” application to see how a map such as that shown in Figure 1 evolves.

Let us see now how this learning occurs. Suppose we have access to a large set of data, in which every data sample consists of three values. A portion of such a set is shown in Table 1.

When constructing a SOM, the number of weights at each node is always set by the algorithm to equal the number of parameters for each sample, so in this example each node stores three weights.

The algorithm proceeds in a series of cycles, as follows.

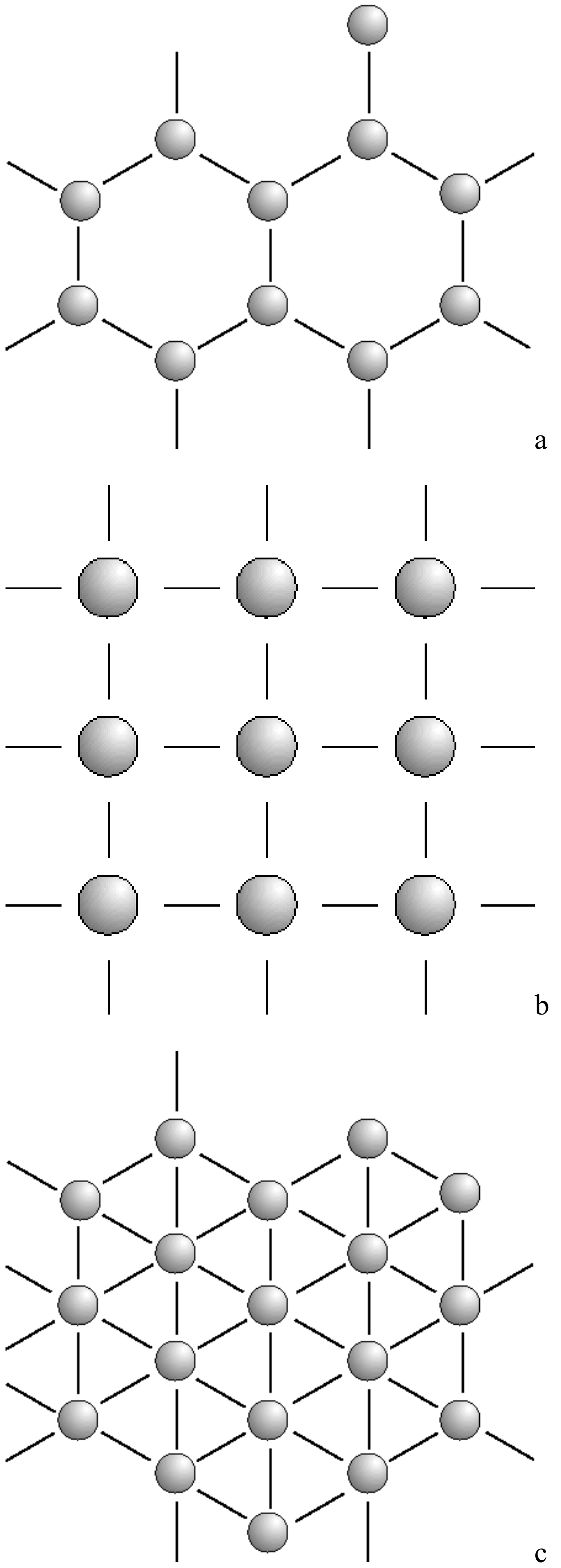

Figure 2. The arrangement of nodes in (a) triangular, (b) square and (c) hexagonal maps.

Table 1. A Portion of a Data Set for Analysis

Sample	Parameter 1	Parameter 2	Parameter 3
S_1	6.7	1.8	2.3
S_2	2.9	0.9	-1.4
S_3	8.2	6.9	11.8
S_4	4.8	4.8	-1.8
S_5	2.1	0	-1.9
S_6	6.7	0.2	0.1
S_7	11.1	0.7	-12.1
...	...	...	...

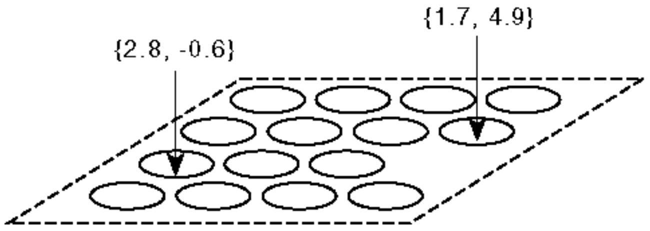

Figure 3. Node weights in a square map.

Initialization. At the start of the calculation, the weights at all nodes are set to random values.

Selection of Sample. A sample is chosen at random from the data set.

Calculation of Deviations. The value of each parameter for the chosen sample is compared in turn with the corresponding weight at the first node and the total difference between sample values and node weights is calculated. Thus, if the weights at the first node were {2.7, 4.1, 0.4} and sample S_5 were chosen, the difference could be calculated as the sum of the absolute magnitude of the differences:

$$d = \sum_{i=1}^{n} |w_i - p_i| = |2.7 - 2.1| + |4.1 - 0 + ||0.4 - (-1.9)| = 7.0$$

in which n is the number of parameter values, w_i the *i*th weight at the current node, and p_i the *i*th parameter value for the current sample. Alternatively, the Euclidean distance could be used:

$$d = \sum_{i=1}^{n} \sqrt{(w_i - p_i)^2}$$
$$= \left\{(2.7 - 2.1)^2 + (4.1 - 0)^2 + (0.4 - (1.9))^2\right\}^{1/2} = 4.7$$

Any "reasonable" method of calculating the difference between sample parameters and node weights may be adopted. This difference is determined for every node in the map.

Selection of the Winning Node. The *winning node* is identified as the one whose weights most closely resemble the sample data; that is, the node for which the difference calculated in step 3 is a minimum.

Modification of Weights at the Winning Node. The node weights at the winning node are now modified slightly to make them more closely resemble the sample data.

$$w_i^* = w_i^*(1 - \alpha) + p_i \alpha$$

In this equation w_i* represents the *i*th weight at the winning node, p_i is the corresponding parameter value for the sample being used, and α is a learning factor whose size determines the extent to which the node weights are adjusted.

Modification of Weights at Neighboring Nodes. The adjustment of node weights is not limited to the winning node, but also extends to nodes nearby. Node weights in the neighborhood of the winning node are modified in the same way as those at the winning node, but by an amount that diminishes with distance from it.

Selection of a Fresh Sample. The learning factor, α, which scales the size of the adjustment to node weights, is now very slightly reduced. This ensures that modifications to the node weights, which may be quite large at the start of the calculation, gradually diminish as the algorithm runs. If the node weights have not converged, the algorithm returns to step 2.

How Might We Use SOMs?

After a large number of passes have been made through the sample data the node weights will converge to stable values; further cycles will produce little further change. Before we consider the steps outlined above in a little more detail, let us consider how we might use the completed map. There are two principle ways in which a SOM can be of value.

Recognition of Dominating Similarity Factors. Figure 1, in which the nodes are drawn in a color determined by the node weights, shows how nodes with similar weights cluster together. This suggests one way in which the map might be used. Consider what happens when we take a sample from the data set and feed it into the map. The algorithm inspects each node to find the one whose weights most closely resemble the sample parameter values and identifies it as the "winning node". We can say that the sample *points* at this node. *Every* sample must point at some node in the map, and we recognize that samples that point to the same node or to nodes that are close together on the map must share similar parameter values.

Now it may seem that the map has not done anything very smart in pointing out to us that several samples in the database are similar. Could we not have recognized this ourselves by inspecting the data and picking out the "similar" samples? If the data are very simple (and very limited in number) this might well be possible; but if each sample consists of perhaps fifteen or twenty parameter values, and there are many samples, searching for similar samples "by eye" is not practicable.

The map, in identifying similar samples, has thus revealed regularities that it may be difficult, or impossible, to spot without the help of some kind of automated procedure.

Once the map has highlighted similar samples for us, we can try to rationalize the clustering of samples. By inspecting the parameter values for samples that cluster together, we may be able to understand what features of the parameters cause the clustering. For example, using SOMs, Barlow showed that the biological activity of histamine H2 agonists could be related to the electrostatic potential around them [4], Zupan found that automotive paint samples could cluster spontaneously into groups which differed in their resistance to weathering, and that this could then be related to the composition of the paint. Cartwright and Kiernan [5] have shown that geochemical and analytical data, such as sample depth and oxygen index, can be used to cluster coal samples according to their potential value in yielding hydrocarbons.

In each case useful deductions can be made by allowing the SOM to cluster data, and then inspecting these clusters to see what characteristics link the different samples together.

Prediction. Suppose we have prepared a complete map through analysis of some suitable database. If a sample is drawn from the database and fed into the map, we know that it will point to a particular node. However, we are not restricted to using samples drawn only from the database that was used to construct the map. If a fresh sample, not contained within the database used to generate the map, is fed into it, the new sample will itself point to some particular node, and by comparing the sample with others that point to the same region, we can gain useful information.

This is particularly helpful if the value of one or more parameters for the new sample is unknown. An interesting example of this technique has been reported by Walker, Cross and Harrison [6] using Growing Cell Structure Networks (GCSN), a close relative of the SOM. They analyzed data from fine-needle aspirates of breast tissue taken from women during the diagnosis of possible breast cancer. A database of 692 samples, each sample consisting of eleven values, was assessed using a GCSN. The map organized the data in such a way that samples derived from patients who tested positive for breast cancer were clustered in one region, and samples derived from patients who tested negative were clustered in another part of the map. By then feeding in data from patients whose diagnosis was uncertain and noting to which region of the map the sample pointed, it was possible to use the trained map as a diagnostic tool. This kind of procedure is typical of the predictive use of the SOM, and is probably its most powerful and useful feature.

Coding of a self-Ordering Map

In this section we consider the coding of a SOM in a little more detail.

Initialization. As a first step the memory of the map must be cleared. For the examples that are used to illustrate this paper the map is square, and of dimension `mapsize` × `mapsize`. The node weights are stored in the three-dimensional array `som_weights[][][]`, in which the first two dimensions are the (*x*, *y*) coordinates of the map nodes, and the third identifies a particular weight at that node. Initially, therefore, we set each entry in this weights array to a randomly chosen value (though it is possible to set them to a single starting value without adversely affecting the calculation). The value is generally chosen to lie within the range of values spanned by the data values, if this is known in advance.

```
//  Initialize the node weights with random values between 1 and mapsize.

mapsize1=mapsize-1;
for (i=0; i<mapsize; i++)
       {
           for (j=0; j<mapsize; j++)
           {
               if (zeroweights)  // Nodes start with identical weights
               {
                   som_weights[i][j][0]= mapsize1/2;
                   som_weights[i][j][1]= mapsize1/2;
                   som_weights[i][j][2]= mapsize1/2;
               }
               else              // Nodes start with random weights
               {
                   som_weights[i][j][0]=1+(mapsize1)*rs.nextFloat();
                   som_weights[i][j][1]=1+(mapsize1)*rs.nextFloat();
                   som_weights[i][j][2]=1+(mapsize1)*rs.nextFloat();
               }
           }
         }
```

Figure 4 shows how the map might look at the start of a calculation where the three weights at each node have been interpreted as RGB values to make it easier to follow the development of the map. Because the weights have been chosen at random, the initial colors of the nodes are also random and the map shows no discernible pattern.

Sample Selection. In this first example, every sample in the database consists of three values. We could start the calculation by generating a database of samples, but it is simpler to create a fresh random data point (x_a, y_a, z_a) whenever a new sample is needed. (On the face of it, it seems remarkable, even counterproductive, to use random data. How can the map find order in entirely random samples? Recall, however, that the ordering that occurs is spatial, that is, samples are spread across the map in such a fashion that samples with very different parameter values are positioned far apart from each other. Ordering is thus possible even of random data!)

```
/* Generate a random data point to feed into the map. */
                   r=280.0*complexity*rs.nextFloat();
                   theta=8.0*rs.nextFloat();
                   phi=3.0*rs.nextFloat();
                   xa=r*Math.sin(theta)*Math.cos(phi);
                   ya=r*Math.sin(theta)*Math.sin(phi);
                   za=r*Math.cos(phi);
```

Calculation of Deviation. The sample data (x_a, y_a, z_a) are now compared, in turn, with the weights at each node. In this program, we sum the absolute differences between the values in the sample and the corresponding weights at the node although, as mentioned above, other ways of calculating this difference exist.

```
diff=Math.abs(xa-som_weights[i][j][0])+Math.abs(ya-som_weights[i][j][1])
             +Math.abs(za-som_weights[i][j][2]);
```

Selection of the Winning Node. The variable `diff` stores the total difference evaluated in step 4.3. If this difference is the smallest found so far in the calculation, the current node is the best yet, and its coordinates (i, j) are stored.

```
/* Find the winning node. Its position is besti, bestj. */

                  besti=0;
                  bestj=0;

/* bestdiff is the smallest difference found in the current cycle between
  the node weights and the current data point.  We start by setting it
  to a number much larger than any difference in the actual calculation
  will be. */

                  bestdiff=500.0;
                  for (i=0; i<mapsize; i++)
                  {
                      for (j=0; j<mapsize; j++)
                      {

/* diff is calculated at this point, as shown in step 3 above.
  If diff is smaller than the best difference so far, update
  bestdiff, and store the position of the winning node. */

                      if (diff<bestdiff)
                          {
                              bestdiff=diff;
                              besti=i;
                              bestj=j;
                          }
                      }
                   }
```

Modification of Weights at the Winning Node. Once the winning node has been found, the weights at this node are updated so that they more closely resemble the data point just used. The following lines accomplish this.

```
som_weights[besti][bestj][0]=som_weights[besti][bestj][0]*(1.0-alpha)+alpha*xa;

som_weights[besti][bestj][1]=som_weights[besti][bestj][1]*(1.0-alpha)+alpha*ya;

som_weights[besti][bestj][2]=som_weights[besti][bestj][2]*(1.0-alpha)+alpha*za;
```

`alpha` is a factor whose size determines how much the node weights are moved in the direction of the data sample.

Modification of Neighborhood Weights. A similar modification is made to the weights of neighboring nodes, with the size of the adjustment determined by the value of `alpha`, diminishing with distance from the node. The coding here is very crude, but this step is fast so sophistication is not required.

```
/*  In many SOMs the "neighbourhood" around each node within
  which weights are updated consists of the entire map at
  the start of the calculation, and diminishes with time.
  In this example the size of the neighbourhood remains
  unchanged, although the size of the weight adjustment
  diminishes as the calculation proceeds. */

alpha=0.3*adjustment;
     mult2=0.06*adjustment;
     for (i=besti-2; i<=besti+2; i++)
        {
           for (j=bestj-2; j<=bestj+2; j++)
           {
     if (i>=0 && i<mapsize && j>=0 && j<mapsize && !(besti==i && bestj==j))
             {
             if (i>besti-2 && i<besti+2 && j>bestj-2 && j<bestj+2)
```

```
                {
                    som_weights[i][j][0]=som_weights[i][j][0]*(1.0-mult1)+mult1*xa;
                    som_weights[i][j][1]=som_weights[i][j][1]*(1.0-mult1)+mult1*ya;
                    som_weights[i][j][2]=som_weights[i][j][2]*(1.0-mult1)+mult1*za;
                }
                else
                {
                    som_weights[i][j][0]=som_weights[i][j][0]*(1.0-mult2)+mult2*xa;
                    som_weights[i][j][1]=som_weights[i][j][1]*(1.0-mult2)+mult2*ya;
                    som_weights[i][j][2]=som_weights[i][j][2]*(1.0-mult2)+mult2*za;
                }
            }
        }
    }
```

Adjustment of the Learning Factor. The learning factor adjustment is changed slightly so that the next change to the node weights will be slightly smaller. In this step `rubble` is a constant defined at the start of the program, and `cycle` counts the number of cycles that have passed. As `cycle` increases, the size of `adjustment` gradually falls.

```
/* The size of the adjustment to the weights each cycle depends
  upon an (arbitrarily-chosen) constant called rubble. It also depends
  on the number of cycles, since we wish the size of the adjustment to
  gradually diminish as the calculation proceeds. The form of
  the expression which defines the variable adjustment is not prescribed,
  and the values 200.0 and 150.0 in the line below, as well as the
  dependence upon the square of the cycle number, can all be modified in
  order to test how convergence of the map is affected. */
  adjustment=rubble/(200.0+(cycle*cycle)/150.0);
```

If the node weights have not converged, the calculation then continues with the selection of a fresh sample.

Figures 4–13 show how the map evolves during a typical run. The gradual development of order from an initially chaotic state is very evident.

Figures 14–18 illustrate the application of SOMs to a similar problem, but one in which the method chosen to display the progress of the algorithm is quite different. In this instance, each data sample consists of just two values, chosen within the range `0-(mapsize-1).` To follow the evolution of node weights, an unusual method of plotting them is chosen. The weights of node (*i*, *j*) are interpreted as *x* and *y* coordinates, and a spot is drawn at the position defined by *x* and *y*. Spots are drawn for every node, and spots corresponding to neighboring nodes are then joined with a line. Thus, in this square array of nodes, the spot whose position is determined by the weights at node (4, 7), for example, is joined to the spots drawn for nodes (3, 7), (5, 7), (4, 6) and (4, 8). Because the node weights are initially chosen at random, the pattern of spots is initially chaotic; as the map is run; however, structure develops in a fashion which is related very directly to the pattern of input data.

The operation of this map can be seen at http://physchem.ox.ac.uk/~hmc/aistuff/squaresompage1.html

How Can the SOM be Used in Chemistry?

SOMs are a means by which we can identify correlations in complex data sets. They, therefore, potentially have value in assessing a wide variety of data in chemistry and in other fields. Gasteiger and Zupan [7, 8] and Goodacre [9] give a range of examples of how one may take advantage of the special properties of these networks, and several of these are appropriate for undergraduate research projects.

The classic application of SOMs has been to the analysis of olive oils. Olive oil is both a commodity product and a high-value consumer product. Large sums of money could be (and have been) made if high-quality, high-priced olive oil is adulterated by lower quality cheaper oil. Like many natural products, olive oil is a complex mixture of chemicals, and identifying the region of origin is not simple. SOMs have been used to tackle the problem of identification with great success, using GC/MS data as input, and more recent studies [10] (Cartwright and George 2000) have extended the method to investigate the origin and oxidative degradation of wine. SOMs could be applied in a similar way to any natural product whose quality varies with region, method of production, etc..

The analysis of IR spectra is well suited to attack by SOMs, in view of the complexity of the spectra, and the ease with which they can be gathered. A later paper in this series will describe the application of a feedforward neural network in their analysis, but SOMs can also be used with success, as they can for the analysis of NMR spectra.

Finally, a general class of problems results from the need to predict chemical activity from molecular structure. The prediction required may be quite general (of the rate of environmental degradation, for example) or more specific (the biological activity in respect of a particular bacterium, perhaps, or the ease with which a particular bond in a molecule is broken). Once again, provided a sufficiently comprehensive database is available, SOMs can be widely applied to this type of problem.

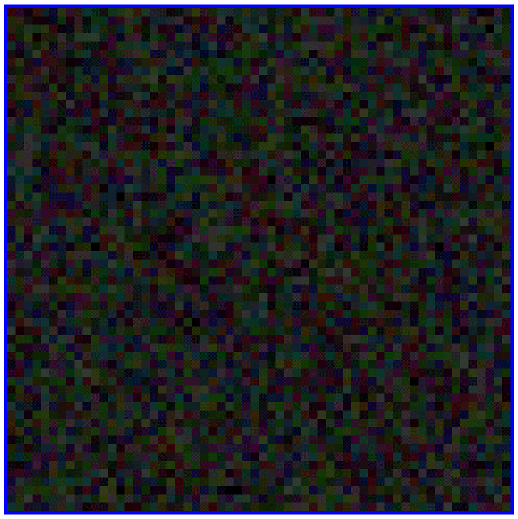

Figure 4. At the start of the algorithm. Because node weights are chosen at random, there is no structure.

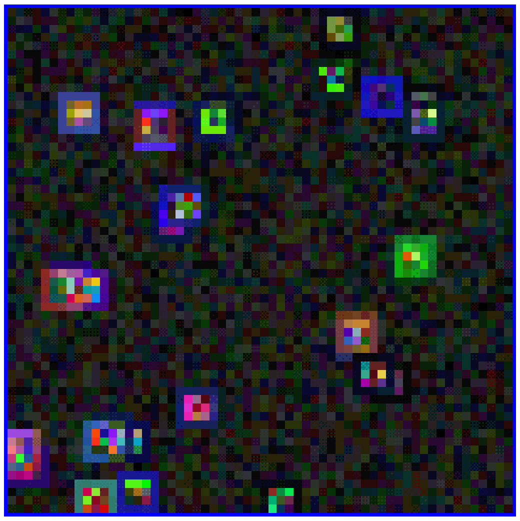

Figure 5. Cycle 11. The highlighted regions show where the first few "winning nodes" have been found.

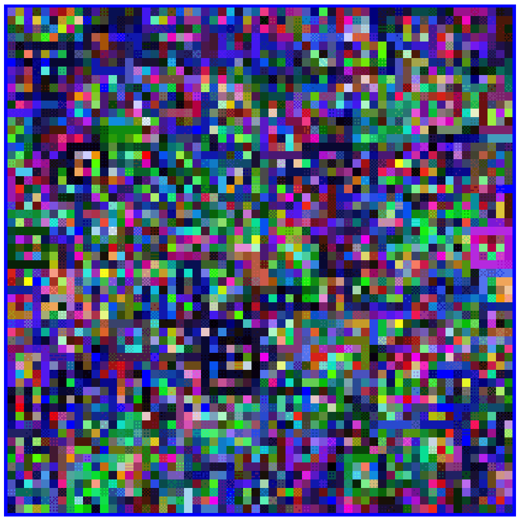

Figure 6. Cycle 202. All nodes have now been modified by the calculation, but still no structure is apparent.

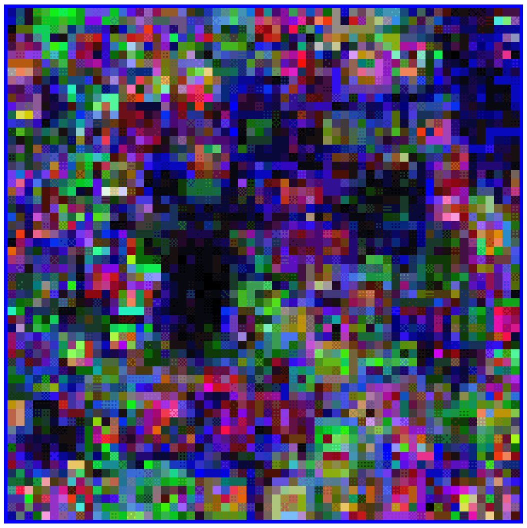

Figure 7. Cycle 4403. Some color separation is starting to appear, notably in the development of dark regions in the center of the map.

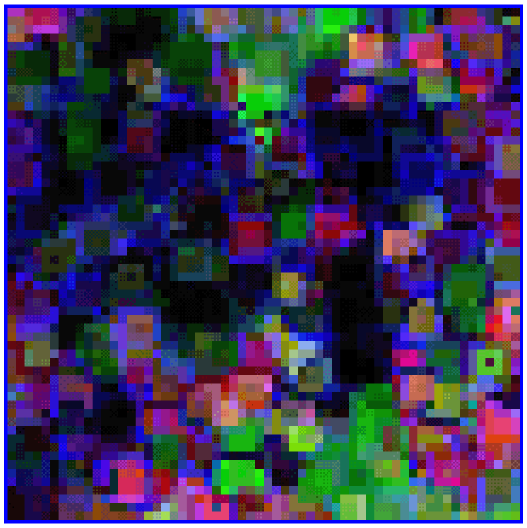

Figure 8. Cycle 5976.

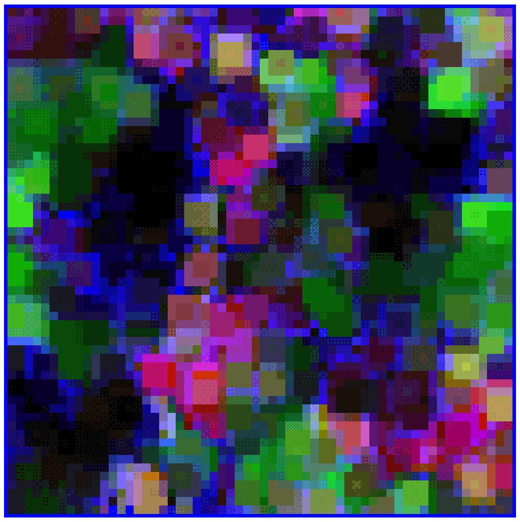

Figure 9. Cycle 8137. Development of the different regions is now becoming pronounced.

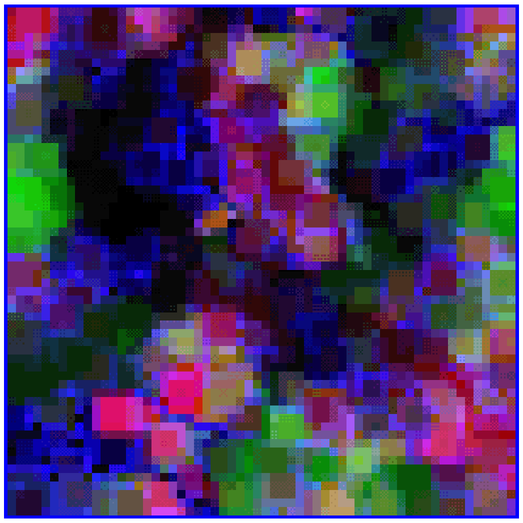

Figure 10. Cycle 9440.

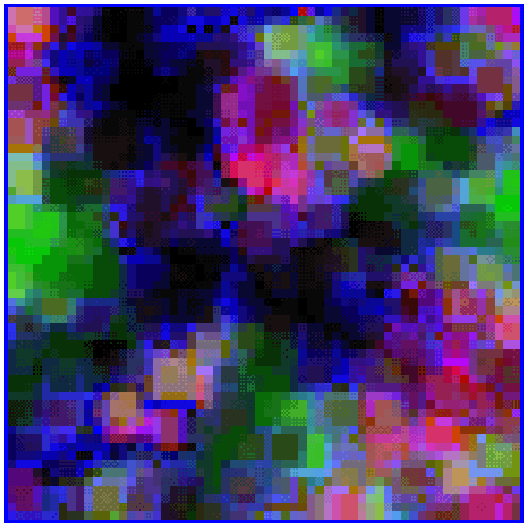

Figure 11. Cycle 11110.

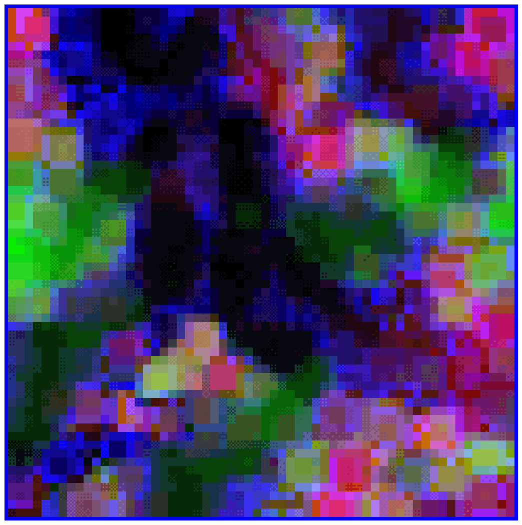

Figure 12. Cycle 13050.

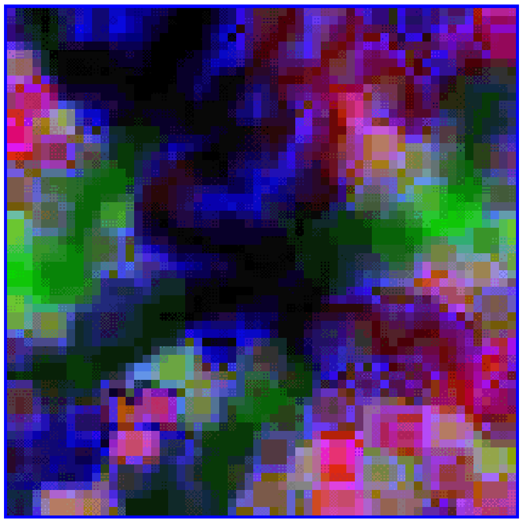

Figure 13. Cycle 20842. The node weights are now essentially stable and the map is complete.

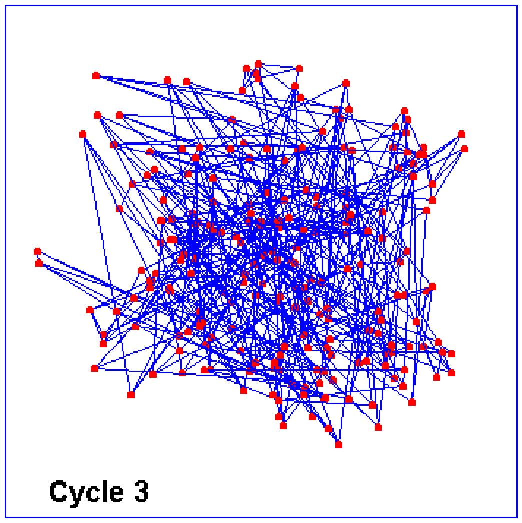

Figure 14. Near the start of the algorithm, the arrangement of nodes, reflecting their weights, is random.

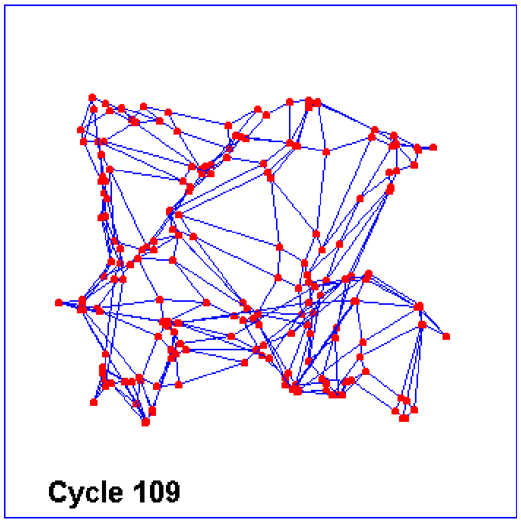

Figure 15. Cycle 109. The network remains very disorganized, but there are clear signs of structure developing.

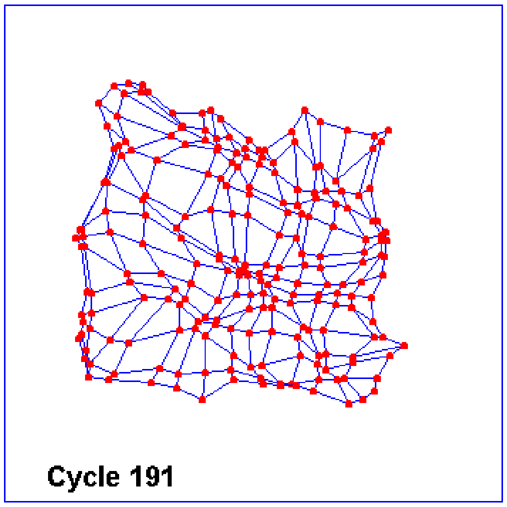

Figure 16. Cycle 191. The structure is becoming clear. Node spacing is now much more even than earlier in the calculation, though the structure remains irregular.

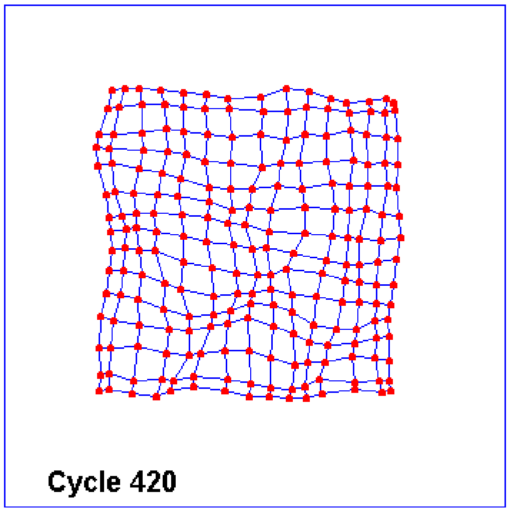

Figure 17. Cycle 420. The final form of the map is becoming well defined.

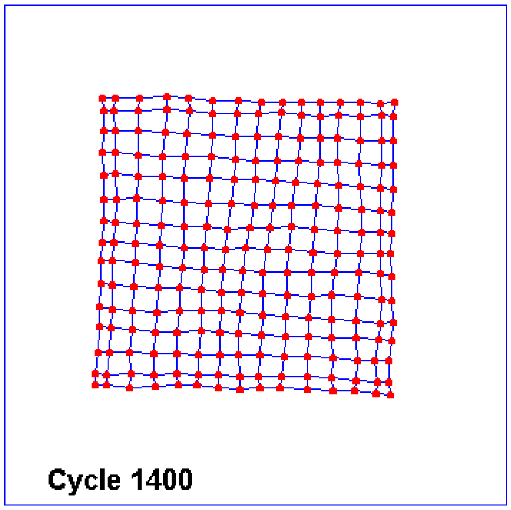

Figure 18. Cycle 1400. The node weights have now almost fully converged.

Supporting Information Available Online

Supporting materials are available from both *The Chemical Educator* web site and from the author's web site located at http://physchem.ox.ac.uk/~hmc/papers/chedr2000a/chedr2000aindex.html.

The URL's given above can be used to access the following material related to this paper:

(a) Online (color) copies of Figures 1 to 18.

(b) HTML pages to run a Java program that illustrates the development of maps of the type shown in Figure 1 and Figures 4–13.

(c) Program listing of the Java applet used to generate Figure 1 and Figures 4–13.

(d) HTML pages to run a Java applet illustrating the development of the map shown in Figures 14–18.

(e) Program listing for the program used to generate Figures 14–18.

References and Notes

1. Malinowski, E .R.; Howery, D. G. *Factor Analysis in Chemistry;* Wiley: New York, 1980.
2. Kohonen, T. Self-organized formation of topologically correct feature maps. *Biol Cybernetics* **1982,** *43,* 59–62.
3. It will be apparent that nodes at the edges of such an array have fewer neighbors than nodes in the interior and therefore are qualitatively different from them. It is often helpful to join opposite edges of the array to form a torus, so that all nodes have the same number of neighbors. We shall in this paper consider only those topologies that are nontoroidal, because these are slightly simpler computationally, but toroidal geometries do not introduce significant complications.
4. Barlow, T. W.; Self-organizing maps and molecular similarity. *J. Mol. Graphics* **1995,** *13,* 24–27.
5. Cartwright, H. M.; Kiernan, K. Assessment of the hydrocarbon potential of oil-bearing rocks using self-organizing maps, in preparation.
6. Walker, A. J.; Cross, S. S.; Harrison, R. F. Visualisation of biomedical datasets by use of growing cell structure networks: a novel diagnostic classification technique. *The Lancet* **1999,** ***354,*** 1518–1521.
7. Gasteiger, J.; Zupan, J. Neural networks in chemistry. *Angew. Chem. Int. Enlg.* **1993,** *32,* 503–527.
8. Zupan, J.; Gasteiger, J. *Neural Networks in Chemistry and Drug Design*. Wiley: New York, 1999.
9. Goodacre, R. Applications of artificial neural networks to the analysis of multivariate data. In *Intelligent Data Analysis in Science;* Cartwright, H. M., Ed.; Oxford University Press: Oxford, UK, 2000, 123–152.
10. Cartwright H. M.; George, J. The use of self-organizing maps in the prediction of wine oxidation, in preparation.