Project 2:
Self-Organizing Map

Dale Patterson
(email: dpatte3@cs.umbc.edu)

April 22, 2006

Contents

1 Self-organizing Map

Kohonen's self-organizing map (SOM) is a 1-layer competitive learning network which topologically structures the output nodes using a predefined network topology. In order to simplify the description in further sections, we will use the following notation: Let p = the number of input vectors, n = the dimension of each input vector and m = the number of nodes in the second layer. The training sample $T = \{i_1, ...i_p\}$ consists of p input vectors of the form $i_p = \{i_{p,1}...i_{p,n}\}$. The network is composed of a m-by-n weight matrix w, such that $w_{i,j}$ denotes the weight of the connection from the j^{th} node in the input layer to the i^{th} node in the second layer. Unlike previous algorithms, i.e. backpropagation in Project 1, each output node i's weight vector $w_i = w_{i,1},...w_{i,n}$ represents a cluster centroid, to which an input vector from T is associated with only if the squared euclidean distance of the input from w_i is minimum. In other words, an input vector i_p is associated with node i iff:

$$\forall j : \sum_{l}^{n}(i_{p,l} - w_{i,l})^2 < \sum_{l}^{n}(i_{p,l} - w_{j,l})^2 \tag{1}$$

We shall refer to the node i having the minimum squared euclidean distance as the winner for the given input. For each presentation of input i_p, the weights in the network are updated using Equation 2 shown below.

$$w_l = \begin{cases} w_l + \eta(i_p - w_l) & \text{if } l \in N_i \\ w_l & \text{otherwise} \end{cases} \tag{2}$$

In Equation 2, η is the learning rate and N_i is defined as the neighborhood of node i, the winner for input i_p. The neighborhood is defined as all nodes within a specified topological distance D from node i. The full specification of neighboring nodes is given in Section 2.3.

2 Description

For this project, the input consists of 2-dimensional vectors representing 60 points randomly spread along a circle of radius 1, with origin at $(0, 0)$. The network uses twelve nodes arranged in the circular ring topology of Section 2.3. That is, $p = 60, n = 2, m = 12$.

2.1 Input Generation

$T = \{i_1, ...i_p\}$, the training sample is a set of uniformly distributed random numbers generated in the range $[-1, 1]$. After generation, each input vector $i_p = \{i_{p,1}, i_{p,2}\}$ is normalized by dividing by $\sqrt{{i_{p,1}}^2 + {i_{p,2}}^2}$. The normalization ensures that the input points lie on the parameter of the unit circle at origin $(0, 0)$.

2.2 Node Vector Generation

Each node i in the second layer, has a 2-dimensional vector w_i which is randomly generated in the same manner as described in Section 2.1. Furthermore, the weight vectors are normalized after each weight update.

2.3 Network Topology

The network topology for this project is a circular ring. Nodes in the output layered are numbered $1...m$. We shall define two output nodes i and j as **adjacent** if and only if the distance between them is one. So, $\forall i, j : i$ and j are adjacent iff $i = j \pm 1$. Additionally, because of the circular topology, node 1 and node m are adjacent. The neighborhood of node i, N_i, is any node (inclusive i) within $D \geq 0$ adjacent nodes of i. Each node is withing a distance $D = 0$ of itself. D is a dynamic parameter, the value of which will be defined later in Section 3.2.

2.4 Visualization

In Section 3.4, several figures are shown of the SOM network at different times. This section describes, the visualization. Each node $1..m$ is depicted as a circle and is assigned a different color as defined in Figure 1. The unit circle is displayed as a yellow line. Input

Figure 1: Colors assigned to individual nodes in the output layer.

points are depicted as stars, and each input point is colored the same color as the node to which it is clustered at the current time. Black lines are also shown connecting adjacent nodes.

3 Report

After finalization, ten sets of training samples were generated. For each set, a network was generated and executed.

3.1 Presentation of Inputs

The sample inputs are randomly permutated for each epoch so that input vectors are never presented in the same order.

3.2 Parameters

As stated in Section 2, the Training sample is a set of 60 2-dimensional input vectors. There are twelve nodes in the output layer. The learning rate, η, and D, the neighborhood radius, are dynamic and follow the following reduction schedules in Equation 3 and Equation 4.

$$\eta(t) = \begin{cases} 0.5 & \text{if } t = 0 \\ \eta(t-1) - 0.005 & \text{for } t = 1...100 \\ 0.005 & \text{otherwise} \end{cases} \tag{3}$$

$$D(t) = \begin{cases} 4 & \text{if } t \leq 1 \\ 3 & \text{if } 1 < t \leq 2 \\ 2 & \text{if } 2 < t \leq 10 \\ 1 & \text{if } 10 < t \leq 20 \\ 0 & \text{if } t > 20 \end{cases} \tag{4}$$

In both equations, t is defined as the current epoch. Equation 4 was chosen because defining the neighborhood radius as 4 for the first two epochs and then three for the next two epochs provided a further randomization of the initial node weights although most networks began to stabilize in a correctly ordered topology once the neighborhood radius decreased to three and never no later than a radius of two. Increasing the rate of reduction in η usually resulted in a final configuration that was less satisfiable because the network quickly reached a certain state from which the nodes did not vary. If the input vectors were a bad sampling of a unit circle, then the final network would not converge on a unit circle.

3.3 Convergence

The networks went through several stages of stabilization in route to convergence. In the first two epochs because of the high neighborhood radius and high learning rate, nodes fluctuated over the perimeter of the circle. Generally during the third or fourth epoch, nodes begin to stabilize in a correct order. That is, each pair of adjacent nodes arranged themselves adjacently to each other with no intervening node between them. During presentations of input vectors, nodes tended to jump over adjacent nodes but the correct order was eventually returned and the cluster of point associated with each node changed dynamically. After 20 to 30 epochs with a neighborhood radius of 0, the networks had usually stabilized such that each node's cluster of input points changed slightly, but the nodes weight vectors ranged over the cluster due to the learning rate. At this point, the network's topology resembled the unit circle. Around 60 epochs, the networks generally converged and the change in node vectors was small. I decided to terminate at 120 epochs which allowed the network 20 epochs of presentations with a distance D of 0 and a learning rate η of 0.005 although after 80 epochs, the networks changed little.

3.3.1 Attempting to Measure Convergence

Due to its random generation, the input vectors played a large role in the final topology of the network. No networks failed to arrange themselves in correct order but some input samples naturally led to a network whos final topology closely resembled a circle while others did not. At some times before termination, I noticed that the network reached a state such that it closely resembled the unit circle. Therefore, I attempted to find an accurate measure of convergence in order to allow for the early termination of SOM in a satisfiable topology.

Partition Error I first used the partition error (pg. 157 in the text):

$$E = \sum_{clusters} \sum_{patterns} \frac{||(pattern - cluster\ centroid)||^2}{number of patterns} \tag{5}$$

We would expect a small E to be unreachable as some cluster may range over a large portion of the circle and thus some patterns in the cluster would be relatively far from the cluster centroid. I found that terminating when $E \leq 0.01$ could in some cases result in an adequate convergence but in other cases, the resulting topology did not accurately resemble the unit circle. This happened when the sample input did not cover the perimeter of the unit circle and some clusters were small. No amount of tinkering with E allowed for a satisfiable convergence and using E as a termination criteria was dropped.

Using Area Since the SOM graphically represented a unit circle, I decided to try the area of the circle as a criteria for termination.

$$E = UnitCircle_{Area} - Network_{Area} \tag{6}$$

where $UnitCircle_{Area} = \pi$. This however did not provide an accurate measure of network convergence because using only twelve nodes, the network area approached an upper bound that was less than π. This led to the question of what is this upper bound. While no estimation of the upper bound was found, the final configuration of an ideal network with twelve nodes could be used to approximate the upper bound. An ideal network would have twelve nodes with each adjacent pair of nodes placed equidistantly from each other. A clock is good example of an ideal network topology, each node 1...12 placed at 1 $O'Clock$...12 $O'Clock$ respectively. The area of this ideal network is 2.9735 and the resulting error measurement is

$$E = 2.9735 - Network_{Area}. \tag{7}$$

Of the ten samples, only one finished with an area less than 2.9 and most terminated early having an $area \geq 2.9$. I decided not to use Equation 7 as a termination criteria however because it implies that the network has prior knowledge of the problem to which it is being applied which is not biologically plausible. Furthermore, this equation limits the SOM algorithm to a small subset of possible problems, namely those in which the input vectors sample a unit circle. Different problems would necessitate the discovery of an ideal topology for each and certain problem like TSP do not have a unique ideal configuration. There are some interesting results however as shown in Figure 2. Figure 2 shows the plot of the network area from epochs 20 - 120 for the sixth sample set of input vectors. We see that in epochs 20 - 40, the network area fluctuates vastly but that by epoch 90, the network area stabilizes and converges.

3.4 SOM Results

This section describes the result obtained from the sixth sample set. Figure 3 shows a graphical display of the SOM at (a) initialization, (b) after 3 epochs, (c) after 10 epochs and (d) final map. I chose sample 6 for the results because the network quickly stabilized. Already after epoch 10, the nodes had neared the general area on the unit circle to which they appear in the final configuration. Results for all ten different samples tested in this project are posted on my website,

`http://www.csee.umbc.edu/~dpatte3/nn/som.html`

including an four .avi files showing the complete progress of the SOM over sample 6.

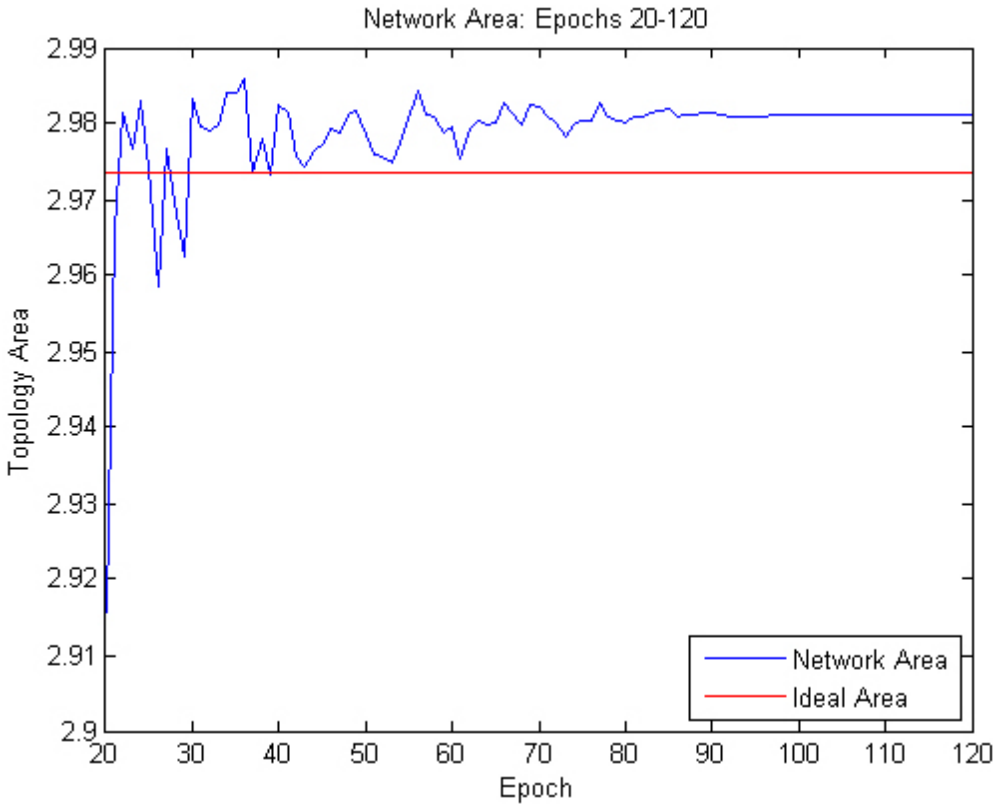

Figure 2: The network area curve relative to the ideal area over epochs 20 - 120.

3.4.1 Final Input Clusters

Below, are listed the set of input vectors $Cluster_{i,j}$ clustered to each node i having weight vector w_i. Using a visual display, the final network topologies can be measured qualitatively by how close they resemble the unit circle. I wanted a better measure of the quality of each node and its associated cluster especially in the cases of bad training samples. We would expect a good final network to have two properties 1) points assigned to a cluster are similar to the cluster centroid and 2) points assigned to one cluster are dissimilar from other cluster centroids. In addition to the points clustered to each node, I have included the mean and standard deviation of the euclidean distance from the points in the cluster to the cluster centroid as well as the mean and standard deviation of the points in a cluster from adjacent centroids: the topologically previous and next cluster centroids.

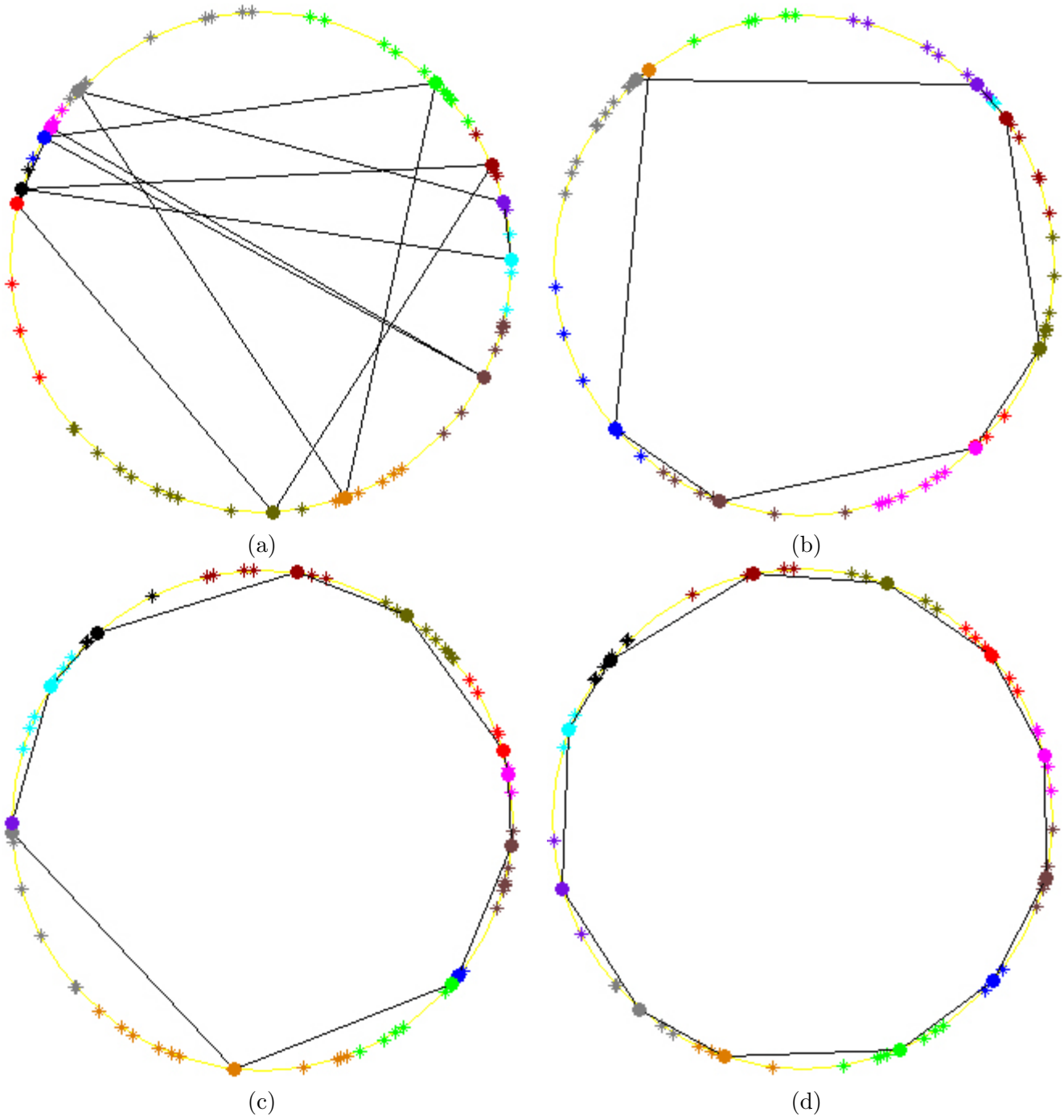

Figure 3: SOM Maps. **(a)** At initialization. **(b)** After epoch 3: $\eta = 0.49$, $D = 3$. **(c)** After epoch 10: $\eta = 0.45$, $D = 2$. **(d)** After termination.

Node 1 $w_1 = (0.7560, 0.6545)$
$Cluster_{1,1} = (0.6925, 0.7214)$
$Cluster_{1,2} = (0.7590, 0.6511)$
$Cluster_{1,3} = (0.8600, 0.5103)$
$Cluster_{1,4} = (0.7404, 0.6721)$
$Cluster_{1,5} = (0.7616, 0.6480)$
$Cluster_{1,6} = (0.7312, 0.6821)$
$Cluster_{1,7} = (0.6499, 0.7600)$
$Cluster_{1,8} = (0.8248, 0.5654)$
Distance From Centroid: Mean = 0.0758, STD = 0.0669.
Distance From Adjacent: Mean = 0.4779, STD = 0.1026.

Node 2 $w_2 = (0.9667, 0.2560)$
$Cluster_{2,1} = (0.9785, 0.2064)$
$Cluster_{2,2} = (0.9342, 0.3569)$
$Cluster_{2,3} = (0.9393, 0.3431)$
$Cluster_{2,4} = (0.9939, 0.1101)$
Distance From Centroid: Mean = 0.0992, STD = 0.0402.
Distance From Adjacent: Mean = 0.4689, STD = 0.1109.

Node 3 $w_3 = (0.9727, -0.2322)$
$Cluster_{3,1} = (0.9712, -0.2382)$
$Cluster_{3,2} = (0.9370, -0.3494)$
$Cluster_{3,3} = (0.9592, -0.2827)$
$Cluster_{3,4} = (0.9655, -0.2602)$
$Cluster_{3,5} = (0.9670, -0.2547)$
$Cluster_{3,6} = (0.9991, -0.0425)$
$Cluster_{3,7} = (0.9819, -0.1893)$
Distance From Centroid: Mean = 0.0669, STD = 0.0663.
Distance From Adjacent: Mean = 0.4745, STD = 0.0926.

Node 4 $w_4 = (0.7647, -0.6444)$
$Cluster_{4,1} = (0.7273, -0.6863)$
$Cluster_{4,2} = (0.7992, -0.6011)$
Distance From Centroid: Mean = 0.0558, STD = 0.0005.
Distance From Adjacent: Mean = 0.4647, STD = 0.0627.

Node 5 $w_5 = (0.3878, -0.9218)$
$Cluster_{5,1} = (0.5316, -0.8470)$

$Cluster_{5,2} = (0.3867, -0.9222)$
$Cluster_{5,3} = (0.3371, -0.9415)$
$Cluster_{5,4} = (0.5620, -0.8271)$
$Cluster_{5,5} = (0.1659, -0.9861)$
$Cluster_{5,6} = (0.4841, -0.8750)$
$Cluster_{5,7} = (0.2961, -0.9552)$
$Cluster_{5,8} = (0.3129, -0.9498)$
Distance From Centroid: Mean = 0.1164, STD = 0.0764.
Distance From Adjacent: Mean = 0.5819, STD = 0.1799.

Node 6 $w_6 = (-0.3102, -0.9507)$
$Cluster_{6,1} = (-0.4176, -0.9086)$
$Cluster_{6,2} = (-0.3635, -0.9316)$
$Cluster_{6,3} = (-0.1215, -0.9926)$
$Cluster_{6,4} = (-0.3304, -0.9438)$
Distance From Centroid: Mean = 0.0967, STD = 0.0752.
Distance From Adjacent: Mean = 0.5436, STD = 0.2034.

Node 7 $w_7 = (-0.6506, -0.7594)$
$Cluster_{7,1} = (-0.5173, -0.8558)$
$Cluster_{7,2} = (-0.7454, -0.6666)$
$Cluster_{7,3} = (-0.5651, -0.8250)$
$Cluster_{7,4} = (-0.6509, -0.7592)$
$Cluster_{7,5} = (-0.7487, -0.6629)$
Distance From Centroid: Mean = 0.1086, STD = 0.0638.
Distance From Adjacent: Mean = 0.4803, STD = 0.1578.

Node 8 $w_8 = (-0.9604, -0.2786)$
$Cluster_{8,1} = (-0.9613, -0.2756)$
$Cluster_{8,2} = (-0.8877, -0.4605)$
$Cluster_{8,3} = (-0.9962, -0.0876)$
Distance From Centroid: Mean = 0.1311, STD = 0.1109.
Distance From Adjacent: Mean = 0.6031, STD = 0.1701.

Node 9 $w_9 = (-0.9334, 0.3589)$
$Cluster_{9,1} = (-0.9098, 0.4150)$
$Cluster_{9,2} = (-0.9574, 0.2887)$
$Cluster_{9,3} = (-0.9285, 0.3713)$

Distance From Centroid: Mean = 0.0495, STD = 0.0320.
Distance From Adjacent: Mean = 0.4793, STD = 0.1835.

Node 10 $w_1 0 = (-0.7716, 0.6361)$
$Cluster_{10,1} = (-0.6973, 0.7168)$
$Cluster_{10,2} = (-0.8271, 0.5621)$
$Cluster_{10,3} = (-0.8301, 0.5576)$
$Cluster_{10,4} = (-0.7060, 0.7082)$
$Cluster_{10,5} = (-0.7591, 0.6510)$
$Cluster_{10,6} = (-0.7925, 0.6099)$
Distance From Centroid: Mean = 0.0751, STD = 0.0384.
Distance From Adjacent: Mean = 0.4956, STD = 0.2005.

Node 11 $w_1 1 = (-0.1957, 0.9807)$
$Cluster_{11,1} = (-0.4382, 0.8989)$
$Cluster_{11,2} = (-0.2202, 0.9754)$
$Cluster_{11,3} = (-0.0372, 0.9993)$
$Cluster_{11,4} = (-0.1961, 0.9806)$
$Cluster_{11,5} = (-0.0751, 0.9972)$
Distance From Centroid: Mean = 0.1125, STD = 0.1038.
Distance From Adjacent: Mean = 0.6019, STD = 0.1630.

Node 12 $w_1 2 = (0.3387, 0.9409)$
$Cluster_{12,1} = (0.1940, 0.9810)$
$Cluster_{12,2} = (0.1934, 0.9811)$
$Cluster_{12,3} = (0.2513, 0.9679)$
$Cluster_{12,4} = (0.5387, 0.8425)$
$Cluster_{12,5} = (0.4938, 0.8696)$
Distance From Centroid: Mean = 0.1572, STD = 0.0472.
Distance From Adjacent: Mean = 0.5193, STD = 0.1664.

3.4.2 Final Analysis

Listed below are the mean and standard deviation of the mean distances of the twelve clusters:

> Distance From Centroid: Mean = 0.0954, STD = 0.0321.
> Distance From Adjacent: Mean = 0.5159, STD = 0.0532.

And the mean and standard deviation of the mean distances of the twelve clusters give the initial weight vectors:

Distance From Centroid: Mean = 0.1975, STD = 0.1802.
Distance From Adjacent: Mean = 1.5429, STD = 0.3623.

The above data show that by termination, the clusters are partitioned such that like points are in clusters whose node has a similar weight vector and are farther from adjacent clusters. This is not a full measurement of the quality of the final network configuration. In a SOM, the dissimilarity of points in one cluster from the centroid of other clusters should increase proportional to the topological distance of the clusters. Figure 4 shows that for sample six,

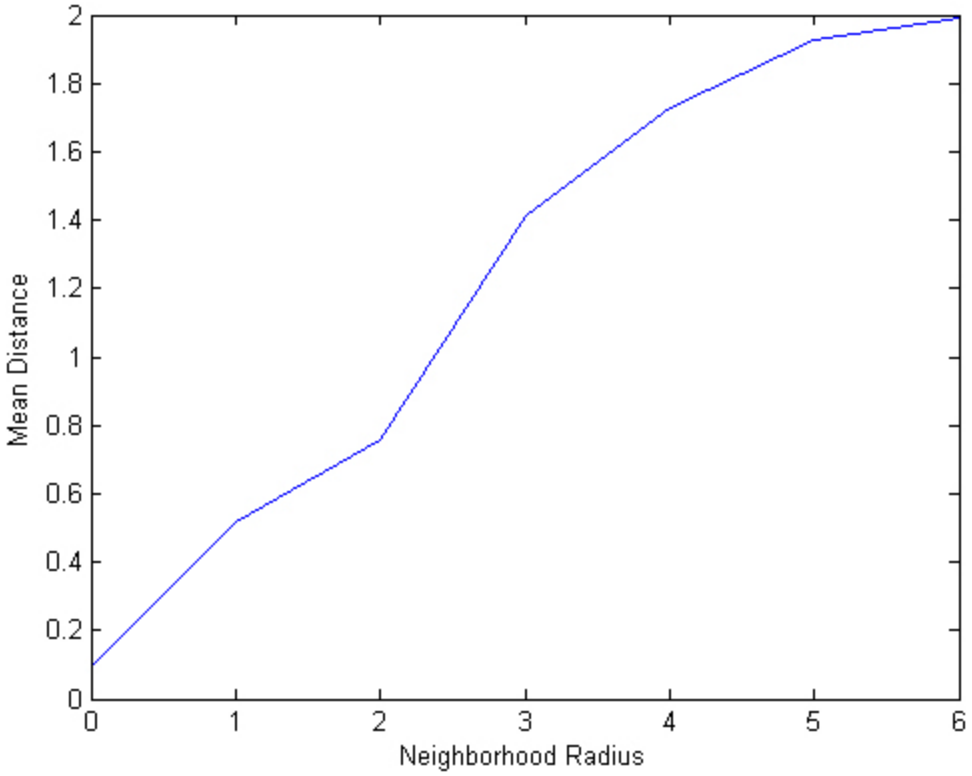

Figure 4: Mean distance of points in each cluster from cluster centroids at neighbor radius = 0..6

the distances from cluster centroids increases as the neighborhood radius increases.

4 Source Code

The SOM algorithm and accompanying programs are written in Matlab 7.1 SP3 and have been executed on Windows XP SP2. For each file, a brief description is given. The code is not discussed because the programs are well documented.

4.1 Program Code

This section lists the main program code.

4.1.1 som.m

Program som takes as input: m=number of nodes, $init\_n$=initial learning rate, dec_n=amount to decrease learning rate by after each epoch, $Input$=a matrix of p input vectors of size n and $File$=name of file to write data from algorithm to which must have a .bin extension. The format of this file can be found in the header documentation and this file can be used to create an animation of the file (Section 4.3.2), plot snapshots of the SOM (Section 4.3.1) or for calculating statistics. It returns w=the final weight vectors. A program, distance.m is required to be in the local directory, or on the path which when given t=iteration, returns D= neighborhood radius for the current iteration. Section 4.2.1 shows the distance function used in this project.

```
% som a self-organizing-map using a circular network topology.
%   w = som(m,init_n,dec_n,init_d,Input,File)
%   returns w, the final weight vector for each node in a circular ring
%   topology, t: the number of iterations and e the squared average distance
%   at termination. Given input a p-by-n matrix of input vectors of dimension
%   n, the network w is a 2-layer net constructed as a m-by-n matrix such
%   that node i's weight vector is w(i,[1-n]). At each presentation of
%   an input vector i(l), the winner w(j) is found and the winner and
%   neighbors (according to the current distance function) are updated.
%   Uses a circular ring topology such that node i is adjacent to node j if
%   i = j+1 or i=j-1 and node 1 is adajacent to node m.
%
%   NOTE: this version of som expects input to be of unit length and as
%   such calculates the winner wj = max(i(l)*w). Furthermore, this som
%   normalizes the weight vectors after each presentation such that they
%   lie on a unit circle.
%
%   Additionally, writes in File (binary format) the weight vectors for
%   each time t=0..termination. The File has the following format:
```

```
%    The header:
%     - a 1-by-3 matrix [p,n,m] where p is the number of input vectors, n is
%       the dimeniondion of the input vectors and m is the number of output
%       nodes.
%     - a p-by-n matrix defining the input vectors
%    For each t=0..termination
%     - a 1-by-4 matrix [t,nt,dt,e,j] where t is the time, nt is the learning
%       rate, dt is the distance e is the squared euclidean distance of
%       the selected input from the winner, and j is the winning node
%     - a m-by-n matrix w, the weight vectors at time t
%
%   Input:
%     m - the number of nodes (output)
%     init_n - the learning rate n at time t=0
%     dec_n - amount to decrease n after each epoch
%     init_d - the distance d at time t=0
%     Input - the input vectors
%     File - the name of a file to write the weights at each time to. This
%      file will be written in binary.
%
%   NOTES:
%    - Expects a function dt = distance(t) to be defined on the path or
%      in the local directory that determines the distance dt given the
%      current time.
%    - This SOM algorithm has been left as general as possible but for the
%      problem description, (a unit circle), the function norm_circle expects
%      2-dimensional inputs.
%    - Under the assumption that the number of nodes in the output layer is
%      relatively small, som uses the following weight update rule:
%        w = w + nt * (il-w) * nj
%      where nj is 1 for a neighbor of winner or zero otherwise so that the
%      effect of the rule can be defined as
%        wj = wj + nt*(il-wj) if j is a neighbor of winner
%        wj = wj              otherwise
%      By doing so, neighbors of the winner do not have to removed from w
%      updated and reinserted. If however, the number of nodes is quite
%      large, it may be less computationally expensive to only calculate
%      il-w for neighbor nodes.
%
% Author: Dale Patterson
```

```
% $Version: 1.4.3 $ $Date: 3.30.06 $
%
function w = som(m,nt,dec_n,dt,Training,File)
% attempt to open the output file
[fid,msg] = fopen(File,'wb');
if fid == -1
    error('project2:cannotOpenReportFile',fmsg);
end

% generate the initial weight vectors
% w is a m-by-n matrix such that w(i,[1-n] is the weight vector of node i
% weights are generated randomly in the range [-1,1] and normalized
[p,n] = size(Training);             % get # of samples and dimension
w = norm_circle(-1 + 2.*rand(m,n)); % and generate w

% since we're using matrix subtraction, we'll make a multidimensional array
% I of size m-by-n-by-p where each I(:,:,p) is a m-by-n matrix A where for
% each row i (i=1..m) in A equals Training(p,:), that is, each input vector
% is repeated m times
I = ones(m,n,p);
for i=1:p
    I(:,:,i) = ones(m,1) * Training(i,:);
end

% write network parameters, input and initial weight vectors
t = 0;                          % start at time t=0
err = Inf;                      % and err as infinity
fwrite(fid,[p,n,m]);            % as a uchar
fwrite(fid,Training,'double');  % use double for precision
fwrite(fid,[t,nt,dt,err,NaN],'double');
fwrite(fid,w,'double');

% loop till computational bounds are exceeded
% we'll terminate after 120 epochs, for this problem, that means 20 epochs
% at d=0 and n=0.005
while t < p*120
    % permutate the input so that there's a different order for each epoch
    i = randperm(p); % i is a random permuatation of 1..p, use i to index I
    err = 0;
```

```
    % loop over the input (an epoch)
    for l=1:p
        % update the time
        t = t + 1;
        il = I(:,:,i(l)); % select an input sample

        % calculate distance, from i to each node
        dist = il-w;                        % distance from i to each node
        [v,j] = min(sum(dist.^2,2));        % j: winner index, v: sq. euc. dist.
        nj = neighbors(j,m,dt) * ones(1,n);% nj: 1s for neighbors, 0s otherwise
        err = err + v;                      % tally the error

        % update weight of winner and neighbors, and normalize
        w = norm_circle(w + nt * dist .* nj);

        % write period parameters and weight vectors to file
        fwrite(fid,[t,nt,dt,v,j],'double');
        fwrite(fid,w,'double');
    end

    % calculate the error;
    err = err / p;

    % update distance and learning rate
    if nt - dec_n > 0
        nt = nt - dec_n;
    end
    dt = distance(t);
end

% clean up
fclose(fid);  % and close the file

%%%% END som     %%%%
%%%% BEGIN SUB PROCEDURES %%%%

% neighbors calculates the neighbors of a given node in a cirular ring topology.
%   n = neighbors(index,numNodes,distance)
%    returns n a column vector of 0 or 1, {0=not a neighbor, 1=neighbor}
%    for the node at index given a vector of numNodes with the given
```

```
%    distance using a circular ring topology
function n = neighbors(i,ns,d)
n = zeros(ns,1);               % initialize a vector of zeros
n(mod((-d:d)+(i-1),ns)+1) = 1; % and set to 1 each neighbor of i (inclusive 1)
```

4.1.2 project2.m

Program project2 is an interface to the SOM algorithm (som.m). It takes as input, p=number of points on the circle to generate as training data, m=number of nodes to use in the second layer and *Report* a string specifying the file to which som will write output to. It returns, pts=the training data and w=the final weight vectors reutrned from som.m.

```
% project2 generates points on a unit circle for use by kohonen's
%    SOM algorithm.
%   [i,w,t,e] = project2(numPoints,numOutNodes,DataFile) generates n
%   umPoints points around the edge of a circle of radius 1 with origin [0,0]
%   and calls som returning i: the sample inputs, w: the final weights, t:
%   the number of iterations at termination and e: the average squared
%   distance at time of termination
%
%   Input:
%     numPoints - number of training points
%     numOutNodes - number of output nodes
%     DataFile - name of file to write data to
%
% Author: Dale Patterson
% $Version: 1.0.1 $ $Date: 3.30.06 $
%
function [pts,w] = project2(p,m,Report)
pts = -1 + 2.*rand(3*p,2);               % generate 3x pts in range [-1,1]
k = randperm(3*p);                       % generate random permutation
pts = norm_circle(pts(k(1:p),:));       % take & normalize p pts to unit circle
w = som(m,0.5,0.005,4,pts,Report); % now get the iterations from som
```

4.2 Helper Functions

This section lists miscelleanous programs used by the other programs.

4.2.1 distance.m

This program is the distance function used by som.

```
% distance(t) returns the distance for time t
function d = distance(t)
if t <= 60
    d = 4;
elseif t <= 120
    d = 3;
elseif t <= 600
    d = 2;
elseif t <= 1200
    d = 1;
else
    d = 0;
end
```

4.2.2 clustered.m

Given the training set I and the weight vectors w, returns the input vectors in I clustered to each node represented by w.

```
% clustered returns a cell array consisting of points associated with nodes.
%   c = clustered(Input,weights)
%   Given weights a matrix of weights numbered 1..m, c is a cell array of
%   size m such that c{j} = the set of input vectors Il from Input that
%   are associated with node j based on the distance of each input in Il
%   from node j.
%   NOTE: this expects the input and weights to be of unit length. It does
%    not explicitly calculate the distance rather uses the maximum dot
%    product
%
% Author: Dale Patterson
% $Version: 1.0.0$ $Date: 3.30.06 $
%
function c = clustered(I,w);
[m,n] = size(w);
c = cell(m,1);
[v,j] = max(I*w',[],2);   % j = node of winner for each pt 1..p
for i=1:m
    c{i} = I(find(j==i),:);
end
```

4.2.3 norm_circle.m

A simple normalization routine which takes p 2-dimenstional data and returns them as a vector of normalized 2-dimensional data.

```
% norm_circle normalizes [x,y] pairs.
%   circle = norm_circle(Unnormalized)
%     circle is a vector of normalized [x,y] pairs such that each [x,y]
%     lies on a circle of radius 1 with origin at [0,0] assuming that each
%     [x,y] is in the range [-1,1]
%
% Author: Dale Patterson
% $Version: 1.0.0 $ $Date: 3.30.06 $
%
function c = norm_circle(U)
% get dimensions for U and report error if n is invalid
[m,n] = size(U);
if n ~= 2
    error('normCircle:invalidInputDimensions', ...
          'Input argument U must be a P-by-2 matrix');
end

c = U;
for i=1:m
    norm = nthroot(c(i,1)^2+c(i,2)^2,2);
    c(i,:) = c(i,:)./norm;
end
```

4.2.4 load_som.m

This program loads the data stored by the som algorithm.

```
% load_som reads in the data saved from
%   [np,i,p,w] = load_som(DataFile) reads the data from DataFile and loads
%   into np: a 1-3 vector for the network paramaters, i: the input a p*n
%   matrix, p: params for each iteration, a 1-by-5-numIterations
%   multidimensional array, w: weights for each iteration, a
%   m-by-n-numIterations multidimensional array
%
%   Input:
%     DataFile - name of file to read data from
```

```
%
% Author: Dale Patterson
% $Version: 1.0.0 $ $Date: 4.03.06 $
%
function [np,i,p,w] = load_som(Data)
% open the data file
[fid,fmsg] = fopen(Data,'r');
if fid == -1
    error('plot_som_error:cannotOpenDataFile',fmsg);
end

% read in network paramaters and input
np = fread(fid,[1,3]);
i = fread(fid,[np(1),np(2)],'double');

p = ones(1,5,120*np(1)+1) .* -1;
w = zeros(np(3),np(2),120*np(1)+1);

% read the data
page = 1;
while ~feof(fid)
    pparams = fread(fid,[1,5],'double');
    if ~isempty(pparams)
        p(:,:,page) = pparams;
        w(:,:,page) = fread(fid,[np(3),np(2)],'double');
        page = page + 1;
    end
end

% close the file
fclose(fid);

% remove unused pages
k = find(p(1,1,:) ~= -1); % k = index of all valid pages
p = p(:,:,k);
w = w(:,:,k);
```

4.3 Visualization

4.3.1 som_plot.m

The program som_plot takes as input a file written by som.m and vector *time* of 1 to 4 elements and subsequently plots a snapshot of the network configuration(s) at the given times. This program was used to generate the snapshots in Figure 3.(a)-(d).

```
% som_plot  plots a snapshot of som.
%   som_plot(DataFile,time) plots a snapshot(s) of the weight vectors
%   for a som network at up to 4 time points.
%
%   Input:
%     DataFile - name of file to read data from
%     Time - a vector of 1 to 4 times to plot weights from, use -1 to specify
%     the final weight vectors. Time must be in ascending order.
%
%
% Author: Dale Patterson
% $Version: 1.0.3 $ $Date: 4.03.06 $
%
function som_plot(Data,time)
% verify time
plots = length(time);
if plots <= 0 || plots > 4
    error('plot_som:timeDimensionInvalid','Time: must be of size 1-4');
end

% open the data file
[fid,fmsg] = fopen(Data,'r');
if fid == -1
    error('plot_som:cannotOpenDataFile',fmsg);
end

% read in network paramaters and input
net_params = fread(fid,[1,3]);
p = net_params(1);
n = net_params(2);
m = net_params(3);
input = fread(fid,[p,n],'double');
% fid is currently at initial weights
```

```
% read period paramaters and weights from file
% som halts at 120 epochs, an epoch is of size p and we need one additional
% for the initial weights and period paramaters
params = ones(1,3,120*p+1) .* -1;
weights = zeros(m,n,120*p+1);

% read the data
page = 1;
while ~feof(fid)
    pparams = fread(fid,[1,5],'double');
    if ~isempty(pparams)
        params(:,:,page) = pparams(1:3); % dont need distance & winner
        weights(:,:,page) = fread(fid,[m,n],'double');
        page = page + 1;
    end
end

% close the file
fclose(fid);

% remove unused pages
k = find(params(1,1,:) ~= -1); % k = index of all valid pages
params = params(:,:,k);
weights = weights(:,:,k);

% now plot the specified snapshots
if plots == 1
    if time(1) == -1
        plot_som2(input,p,n,m,params(:,:,end),weights(:,:,end));
    else
        t = time(1) + 1;
        plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    end
elseif plots == 2
    subplot(1,2,1);
    t = time(1) + 1;
    plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    subplot(1,2,2);
    if time(2) == -1
```

```
        plot_som2(input,p,n,m,params(:,:,end),weights(:,:,end));
    else
        t = time(2) + 1;
        plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    end
elseif plots == 3
    subplot(2,2,1);
    t = time(1) + 1;
    plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    subplot(2,2,2);
    t = time(2) + 1;
    plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    subplot(2,2,3);
    if time(3) == -1
        plot_som2(input,p,n,m,params(:,:,end),weights(:,:,end));
    else
        t = time(3) + 1;
        plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    end
elseif plots == 4
    subplot(2,2,1);
    t = time(1) + 1;
    plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    subplot(2,2,2);
    t = time(2) + 1;
    plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    subplot(2,2,3);
    t = time(3) + 1;
    plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    subplot(2,2,4);
    if time(4) == -1
        plot_som2(input,p,n,m,params(:,:,end),weights(:,:,end));
    else
        t = time(4) + 1;
        plot_som2(input,p,n,m,params(:,:,t),weights(:,:,t));
    end
end

% plot_som2(input,p,n,m,params,weights)
%  plot the som snapshot
```

```
function plot_som2(input,p,n,m,params,weights)
t = 0:pi/20:2*pi;          % plot a yellow circle of
plot(sin(t),cos(t),'y');   % radius 1 @ origin [0,0]
axis([-1.1,1.1,-1.1,1.1]); % move the axis out from edge of the circle
axis square;               % and make x and y axis of same length
set(gca,'XTick',0:-1);     % set x-axis ticks off
set(gca,'YTick',0:-1);     % and y-axis ticks off

% colors is a list of colors for each node and associated points, clusters
% holds the points assocatiated with each node
colors =  [1 0 0;                 % 1 red
           1 0 1;                 % 2 magenta
           0.451 0.2627 0.2627;   % 3 brown
           0 0 1;                 % 4 blue
           0 1 0;                 % 5 green
           0.8706 0.4902 0;       % 6 orange
           0.5 0.5 0.5;           % 7 gray
           0.4784 0.06275 0.8941; % 8 purple
           0 1 1;                 % 9 cyan
           0 0 0;                 % 10 black
           0.6 0 0;               % 11 maroon
           0.4 0.4 0];            % 12 army green

% find and color plot the point clusters according to initial weight
clusters = clustered(input,weights);

hold all
    for i=1:m
        if ~isempty(clusters{i})
            points(i) = plot(clusters{i}(:,1),clusters{i}(:,2),'*', ...
                             'MarkerEdgeColor',colors(i,:), ...
                             'MarkerFaceColor',colors(i,:), ...
                             'MarkerSize',6);
        end
    end
hold off

% plot weights and adjacent lines
lw = [weights;weights(1,:)]; % connect weights(1) to weights(end)
hold all
```

```
    plot(lw(:,1),lw(:,2),'-k'); % plot lines seperate from
    for i=1:m
        plot(weights(i,1),weights(i,2),'o', ...
             'MarkerEdgeColor',colors(i,:), ...
             'MarkerFaceColor',colors(i,:));
    end
hold off

% set the title
title(['epoch: ',num2str(floor(params(1)/p)), ...
       '  \eta:',num2str(params(2)), ...
       '  d:',num2str(params(3))]);
```

4.3.2 som_animate.m

The program som_animate animates the progress of a SOM as recorded in *DataFile*. The optional argument *wait* (the default of which is 0.05) specifies the pause period between drawings of sequential time periods.

```
% som_animate  animates the progress of a ring-SOM on 2 dimensional
%    data points.
%   som_animate(DataFile,wait) plots the progress of a SOM
%   algorithm given the data in DataFile
%
%   Input:
%     DataFile - name of file to read data to
%     pause - time to pause for each iteration
%
% Because of the extensive redrawing required, the following may not 'flow'
%
% Author: Dale Patterson
% $Version: 2.0.1 $ $Date: 3.30.06 $
%
function som_animate(Report,wait)
% initialize defaults if any
if nargin < 2
    wait = 0.05;
end

% PLOT the basic figure and set up the display
```

```
t = 0:pi/20:2*pi;          % plot a yellow circle of
plot(sin(t),cos(t),'y');   % radius 1 @ origin [0,0]
axis([-1.1,1.1,-1.1,1.1]); % move the axis out from edge of the circle
axis square;               % and make x and y axis of same length
set(gca,'XTick',0:-1);     % set x-axis ticks off
set(gca,'YTick',0:-1);     % and y-saxis ticks off

% open the file
fid = fopen(Report,'r');

% read the network paramaters and input points
net_params = fread(fid,[1,3]);
p = net_params(1);
n = net_params(2);
m = net_params(3);
input = fread(fid,[p,n],'double');

% colors is a matrix of colors one for each output node in the net,
% nodes is a vector to store the handles for each node, points is
% vector to store handles for each point and clusters is a cell array
% of points currently associated with each node
colors =  [1 0 0;                 % 1 red
           1 0 1;                 % 2 magenta
           0.451 0.2627 0.2627;   % 3 brown
           0 0 1;                 % 4 blue
           0 1 0;                 % 5 green
           0.8706 0.4902 0;       % 6 orange
           0.5 0.5 0.5;           % 7 gray
           0.4784 0.06275 0.8941; % 8 purple
           0 1 1;                 % 9 cyan
           0 0 0;                 % 10 black
           0.6 0 0;               % 11 maroon
           0.4 0.4 0];            % 12 army green
nodes = zeros(m,1);   % handles for node plots
points = zeros(m,1);  % handles for input plots

% read the first period paramaters and initial weights
pparams = fread(fid,[1,5],'double');
w = fread(fid,[m,n],'double');
```

```
% find and color plot the point clusters according to initial weight
clusters = clustered(input,w);

hold all
    for i=1:m
        if ~isempty(clusters{i})
            points(i) = plot(clusters{i}(:,1),clusters{i}(:,2),'*', ...
                            'MarkerEdgeColor',colors(i,:), ...
                            'MarkerFaceColor',colors(i,:), ...
                            'MarkerSize',6);
            set(points(i),'EraseMode','xor','MarkerSize',6);
        else
            points(i) = NaN;
        end
    end
hold off

% plot initial weights
% we have to plot the weights as two seperate graphs: lines and nodes
% because the lines connect a node to its adjacent nodes, the first and
% last node are connected, but when plotting the node markers we only
% want one node
lw = [w;w(1,:)];                    % connect w(1) to w(end)
hold all
    h = plot(lw(:,1),lw(:,2),'-k'); % plot lines seperate from
    set(h,'EraseMode','xor','MarkerSize',2);
    for i=1:m
        nodes(i) = plot(w(i,1),w(i,2),'o', ...
                        'MarkerEdgeColor',colors(i,:), ...
                        'MarkerFaceColor',colors(i,:));
        set(nodes(i),'EraseMode','xor','MarkerSize',8);
    end
hold off

% set the title
title(['epoch: ',num2str(floor(pparams(1)/p)), ...
       '  \eta:',num2str(pparams(2)), ...
       '  d:',num2str(pparams(3))]);

% loop over and plot the progress of the weights in the file
```

```
while ~feof(fid)
    pparams = fread(fid,[1,5],'double');
    if ~isempty(pparams)
        % read the next weight entry
        w = fread(fid,[m,n],'double');

        % find and color plot the clusters
        clusters = clustered(input,w);
        for i=1:m
            if ~isempty(clusters{i})
                if ~isnan(points(i))
                    set(points(i),'XData',clusters{i}(:,1), ...
                                  'YData',clusters{i}(:,2));
                 else
                    hold on
                    points(i) = plot(clusters{i}(:,1),clusters{i}(:,2),'*', ...
                            'MarkerEdgeColor',colors(i,:), ...
                            'MarkerFaceColor',colors(i,:), ...
                            'MarkerSize',6);
                    set(points(i),'EraseMode','xor','MarkerSize',6);
                    hold off
                end
            end
        end

        % plot the nodes
        lw = [w;w(1,:)];
        set(h,'XData',lw(:,1),'YData',lw(:,2));
        for i=1:m
            set(nodes(i),'XData',w(i,1),'YData',w(i,2));
        end

        % set the title
        title(['epoch: ',num2str(floor(pparams(1)/p)), ...
               '  \eta:',num2str(pparams(2)), ...
               '  d:',num2str(pparams(3))]);

        drawnow;
        pause(wait);
    end
```

```
end
fclose(fid);
```