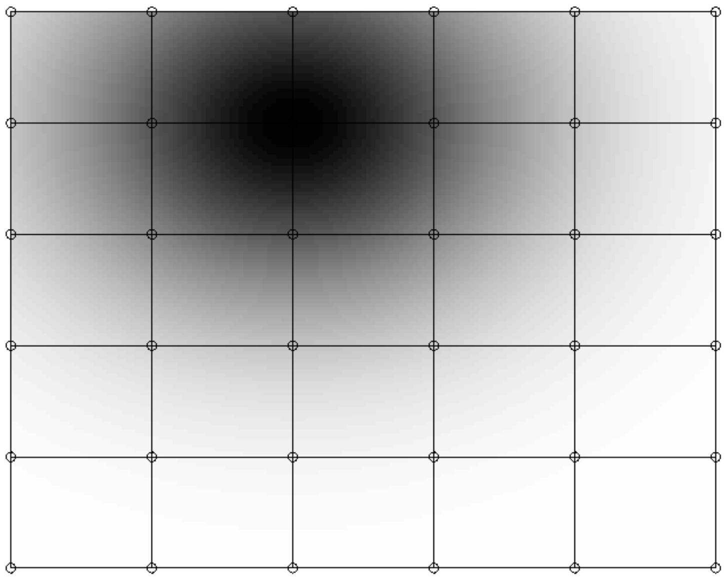

Figure 1: A rectangular array of cells. The grayscale value corresponds to how much the cells are responding to a given pattern. The winning cell is at position $(3, 1)$

Kohonen's Self Organizing Map (2018)

Kohonen's Self Organizing Map (SOM) is important in several ways. The first is that the cluster centers *self-organize* in such a way as to mimic the density of the given data set, but the representation is constrained to a preset structure. We'll see how that works later. Secondly, Kohonen is convinced that this map is a simple model on how neurons in the brain can self-organize. To read more about this, see Kohonen's book [**?**]. Thirdly, the general principles of using this map can be generalized to solve other problems. Again, to read more about this, consult Kohonen's book. We will focus here on the clustering aspects of the SOM.

In biological arrays of neural cells, if one cell is excited, it will dominate the array's response to a given signal. Nearby cells may give a weak response, while cells that are far away give no response at all. One can see this "on-center, off-surround" concept in arrays of visual cells, for example. In Figure 1, we illustrate this concept that is also known as **competition**. Here we see a rectangular array of cells. In this case, the winning cell is at the $(3, 1)$ position. Its *receptive field* is approximately 1.5 units. Outside that field, the cells are not responding.

The key difference in moving from the k-means to the SOM is the idea that each cluster will have **two** representations. One representation of a cluster center will be in the data, usually in $\mathbb{R}^n$, as before. The new twist is that each cluster center ALSO belongs to some "topological structure", like an array of cells.

Typically, cells are arranged in a rectangular or hexagonal grid- although this is not a necessity- they may represent points on a line, or a sphere, or a torus, etc. The important point here is that, whatever the structure, **it is possible to obtain a metric between cluster centers in the array**. These metrics define the relationship between the cluster centers, and is what we mean by "topology".

Definition: Denote the distance between centers i and j using this metric as: $d_{\mathcal{I}}(i,j)$

Before we continue, let's examine a few things more carefully.

Defining the Receptive Field

Suppose we have one cell that's fixed- we'll label it as cell w (later, this will stand for the "winning" cell). The receptive field around the cell in the topological structure will then be defined as the Gaussian,

$$\exp\left(\frac{-d_{\mathcal{I}}^2(i,w)}{\lambda^2}\right)$$

This comes to us from the "normal distribution", which, if x is a scalar value, is typically defined (without the normalizing constant) as:

$$f(x) = \mathrm{e}^{-(x-\mu)^2/2\sigma^2}$$

This has a maximum value of 1 where $x = \mu$ (the mean of the normal distribution), then drops off quickly as x gets farther from μ. How fast the drop off occurs is the function of the standard deviation σ. For a plot with zero mean and unit standard deviation (unscaled), see Figure 2.

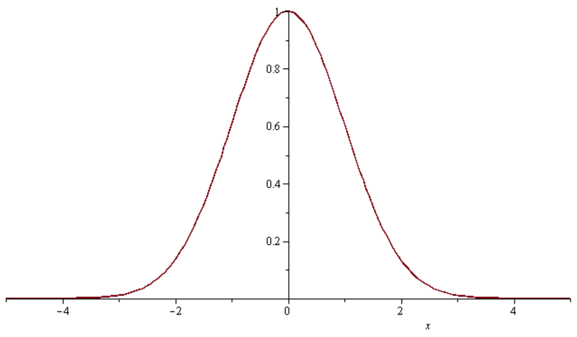

Figure 2: The unscaled normal distribution, $\mathrm{e}^{-(x-\mu)^2/2\sigma^2}$, is used as a model for our receptive field in the SOM.

How will the Algorithm Proceed?

To begin, we'll randomly initialize the centers in the data (similar to the beginning of $k-$means). We will then slowly update the location of the centers using both the location of the centers in the data, AND the location of the centers in the topological array.

We'll do that by taking one data point at a time. The cluster center closest to the data is the winner, and is moved towards the data point. To do that, notice that the line:

$$\boldsymbol{c} = \boldsymbol{c} + \epsilon \cdot (\boldsymbol{x} - \boldsymbol{c})$$

will move $\boldsymbol{c}$ closer to $\boldsymbol{x}$. The other centers are then moved towards the point with an attractive force that is proportional to the metric on the topological array.

Hopefully, the algorithm will "converge" in such a way that the topological structure will unfold itself across the data. Here are three examples of topological structure. In the first function, `topline`, we use a line of cells (and the distance metric reflects that). The second example uses a circle (`topcircle`), and lastly we use an $n \times n$ square grid of values (`toparray`).

```
function distout=topline(i,j,n)
% For a line, cell indices i, j (out of n total) can be thought of as
%  integers on the line.
distout=abs(i-j);
end

function distout=topcircle(j,k,n)
% On the unit circle divided into n arcs, the distance between the jth and
% kth point can be defined as the angle in [0, 2*pi] between the two
% points.
temp=abs(j-k);
t=min(temp,abs(1-temp));
distout=t*2*pi/n;
end

function distout=toparray(j,k,n)
%  Assume we're on an nxn square lattice. Then j and k can be integers from
%  1 to n^2.  The metric is the taxicab metric.

[X,Y]=meshgrid(1:1:n,1:1:n);
Xt=X(:);
Yt=Y(:);
distout=abs(Xt(i)-Xt(j))+abs(Yt(i)-Yt(j));
% Using the 2-norm:
%distout=norm([Xt(i), Yt(i)]-[Xt(j), Yt(j)]);
```

Others are possible of course- How about a sphere, or a torus? Yes- But in the implementation in Matlab, we normally only use a two dimensional hexagonal array (useful for visualizing high dimensional data).

Here we would note that, unlike k−means, the SOM algorithm does not have a known quantity that it minimizes. It's always a good idea to keep an eye on the algorithm as it progresses.

Before getting to the specifics of the algorithm, we'll also need a method of updating the parameters as training progresses. We'll use the power method that we introduced earlier. If α is the parameter of interest, then the value of α at time step k is given as:

$$\alpha(k+1) = \alpha_i \left(\frac{\alpha_f}{\alpha_i}\right)^{\frac{k}{tmax}} \tag{1}$$

Where we set the initial and final values of α as α_i, α_f, respectively, and `tmax` is the maximum number of iterations.

With that, let's not examine the SOM algorithm:

- **Kohonen's SOM Algorithm:**
  - Initialize centers, $\epsilon_{i,f}$, $\lambda_{i,f}$, `tmax`
  - Choose a data point $\boldsymbol{x}$ at random from X.
  - Find the closest center, $\boldsymbol{c}_w$. Call this the winning center.
  - Update all centers. If we're currently at iteration k, then:

    $$\boldsymbol{c}^{(i)}(k+1) = \boldsymbol{c}^{(i)}(k) + \epsilon(k) \cdot \exp\left(\frac{-d_{\mathcal{I}}^2(i,w)}{\lambda^2(k)}\right)\left(\boldsymbol{x} - \boldsymbol{c}^{(i)}(k)\right)$$

  - Update ϵ, λ according to Equation (1).
  - Repeat until a stopping criterion has been attained (usually when `tmax` has been reached).
- **Training Notes:**
  - Initially, use large values of λ (about half the diameter of the set). This strongly enforces the *ordering* of the cells.
  - Small values of λ "relaxes" the ordering, and allows the cluster centers to spread out amongst the data set. The two steps are called the ordering phase, followed by the training phase.
  - Stopping criteria: There are several alternatives. One method is to stop if the centers are no longer changing significantly. Matlab will simply stop after the preset number of iterations.

SOM in Matlab's "Deep Learning Toolbox"

As of 2018, Matlab's neural nets toolbox is being renamed the "Deep Learning Toolbox". In it are some built-in options that we'll explore here for building, training and visualizing SOMs.

There are some things to consider before you use the algorithm:

- What topology (and how many neurons) do you want to use? Matlab provides you with three options: `gridtop`, `hextop`, and `randtop`. All three can be defined in any number of dimensions, although of course, we'll only be able to see the first few. Here is a sample command with a visualization for a hexagonal grid in three dimensions using $4 \times 3 \times 2$ neurons.

```
pos=hextop([4,3,2]);
plotsom(pos);
```

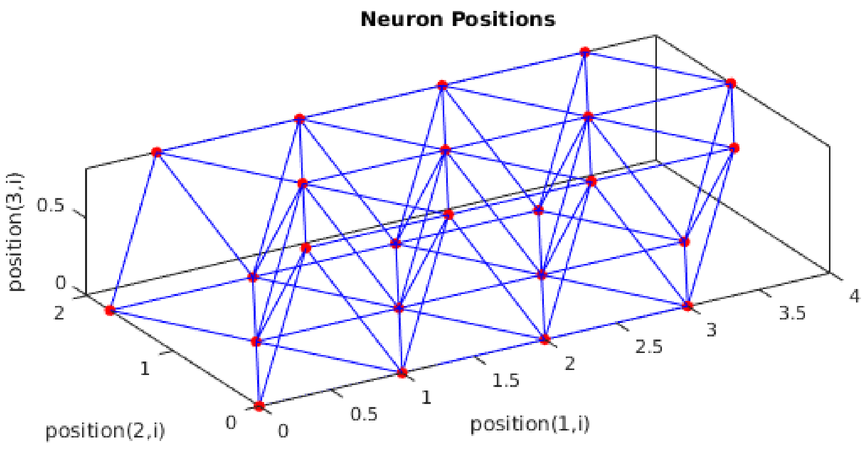

Figure 3: Three dimensional hexagonal topology for the SOM.

- Which distance function should be used? Matlab gives several alternatives:
  - `dist` is the standard Euclidean distance.
  - `linkdist`. See the Matlab help file.
  - `mandist`, or "Manhattan" metric, also known as the "taxicab" metric.
  - `boxdist` is a layer distance function that is commonly used with the grid topology. Its probably easiest to visualize it as below in Figure 4.

Kohonen's Map Example:

Here's a very short example, followed by some discussion of Matlab's training parameters.

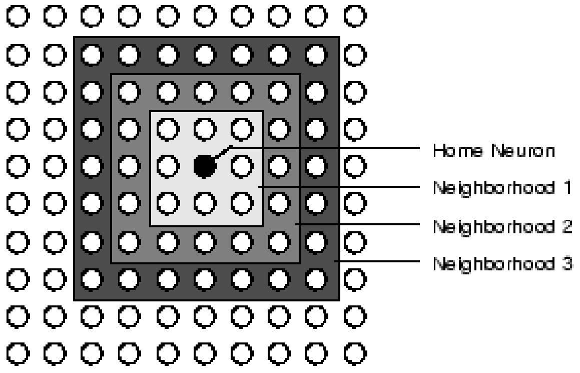

Figure 4: The "box distance" metric. Going from the black neuron in the middle, every neuron in the lightest gray area is one unit away. The next darker gray, every element is two units away, and so on. (Image from the Deep Learning Toolbox).

```
x = simplecluster_dataset;  %2 x 1000, 4 "clusters"
net = selforgmap([8 8]);    %Default topology is hex
net = train(net,x);
y = net(x);                 %Output is 64 x 1000
classes = vec2ind(y);       %Converts to a 1 x 1000
plotsompos(net,x);          %Plot the clusters and data
plotsomtop(net);            %Plot the topology.
```

The 64 centers that we have created have positions in the data that are stored in `net.iw1,1`. A distance matrix is also stored- When we plot, we only connect centers whose distance is 1. This array is in `net.layers1.distances`.

To classify new data points in a data set X, we can type:

```
A=sim(net,X);
```

An important point to note as we go along- Unlike the $k-$means algorithm, the SOM will attempt to "mimic" the density of the data. In fact, there are several papers that exploit this feature to get estimations of the density using the SOM.

Project: Taxonomy

In this project, we use Kohonen's Map to visualize high dimensional data in two dimensions. We will compare this to Principal Components Analysis (PCA).

What's this project about?

In many applications, we are given high dimensional data that we would like to visualize in two dimensions. In this project, the goal of the clustering algorithm is to see which groups of data points belong together - That is, how is the data sitting in that high dimensional space.

One application that we look at here is animal taxonomy, where we group animals together based on their physical characteristics.

Description of the data

The script file `taxonomy.m` defines a set of labels and data corrresponding to a taxonomy of animals- That, 13 animals with 16 characteristics. Calling the script will load a matrix X that is 13×16 with entries either 0 or 1. Each column represents characteristics of one animal, 0 means that characteristic is not present, 1 means that the characteristic is present. The characteristics are (in order):

- Is small, medium, big (first three entries)
- Has 2 legs, 4 legs, hair, hooves, mane, feathers (next 6 entries)
- Likes to fly, run, fly, swim (next 4 entries)

The animals (in order) are: Dove, Hen, Duck, Goose, Owl, Hawk, Eagle. Fox, Dog, Wolf. Cat, Tiger, Lion. Horse, Zebra, Cow. (Grouping with periods was for clarity).

Also after running the script, you should find the variable `G`. If you type `G{1}`, Matlab will return string one, which is `Dove`. This will help to identify the cluster centers in the plots below.

NOTE: Cell arrays are handy to use if you want to store a series of vectors that are not of the same size. They can also be used to store other types of data, and by using strings, one can also use a cell array to index a family of functions (by their m-file names).

Project, Part I: For the 16 data points in $\mathbb{R}^{13}$, project the data to the best two dimensional space. On the two dimensional plot of the 16 data points, label them according to the animal name.

In Matlab, you can plot a text string at coordinates (x, y) by using the `text` command. For example, in the taxonomy file, there is a string of labels named G. Then the following command would plot the labels in random places:

```
x=rand(16,1);
y=rand(16,1);
text(x,y,G);
```

Project, Part II: Use Matlab's SOM commands to map the 16 data points onto a 10×10 rectangular array of neurons. Plot the resulting classifications on the rectangular grid.

Neural Gas

As a reminder, the k-means algorithm performs a simple clustering of the data, putting cluster centers into the data. Kohonen's SOM adds another element, a topological structure on the cluster centers. This is useful if we have a topology in mind (i.e., a rectangular grid for two dimensional visualization of high dimensional data). What if we don't know what topological structure to use?

This question can be re-stated as: Find a *topology preserving* clustering.

Definition: A clustering is said to be **topology preserving** if it maps neighboring cells from the topology to neighboring clusters in $\mathbb{R}^n$, and neighboring data points in $\mathbb{R}^n$ to neighboring cells in the topology.

In Figure 5, we see three topologies mapped to the data, which is a uniform distribution of points in $\mathbb{R}^2$. In the first picture, the topology of the cells is a one dimensional set. In this situation, the cluster mapping is not topology preserving, because neighboring cells in the topology are not adjacent in the plane (the second point and the second to the last point on the line should be far apart, but are mapped close together).

In the second situation, we have a three-dimensional topology mapping to the plane. In this case, neighboring data points in $\mathbb{R}^2$ are mapped to non-neighboring cells in the topology. Only in the third picture do we see that both parts of the topology preserving mapping are satisfied.

The Neural Gas Algorithm [**?**, **?**, **?**] was constructed to do away with an *a priori* topological structure. It will build the topology as it clusters. To do that, we'll need to know how to connect clusters.

Definition: To define the topology, we need to construct a **Connection Matrix**, M, where

$$M_{ij} = \begin{cases} 1 & \text{If cell i connected to j} \\ 0 & \text{Otherwise} \end{cases}$$

so that M, together with the center positions in $\mathbb{R}^n$, form the cluster topology.

Critical to the ability of the SOM to "unfold" into the data was the metric $d_{\mathcal{I}}(i,w)$. We'll need something else to take its place.

The Neural Gas metric

As before, define w as the index of the closest center to a given data point $\boldsymbol{x}$. Sort the centers according to their distances to $c^{(w)}$, and put the ordered indices into a vector, $V = \{w, i_1, i_2, \ldots, i_{k-1}\}$. Therefore, $V(k)$ represents the index of the k^{th} closest center to $C^{(w)}$. Then:

$$d_{ng}(i,w) = k - 1$$

where $V(k) = i$. So, $d_{ng}(i,w)$ counts how many centers are closer to w than center i is.

Example:

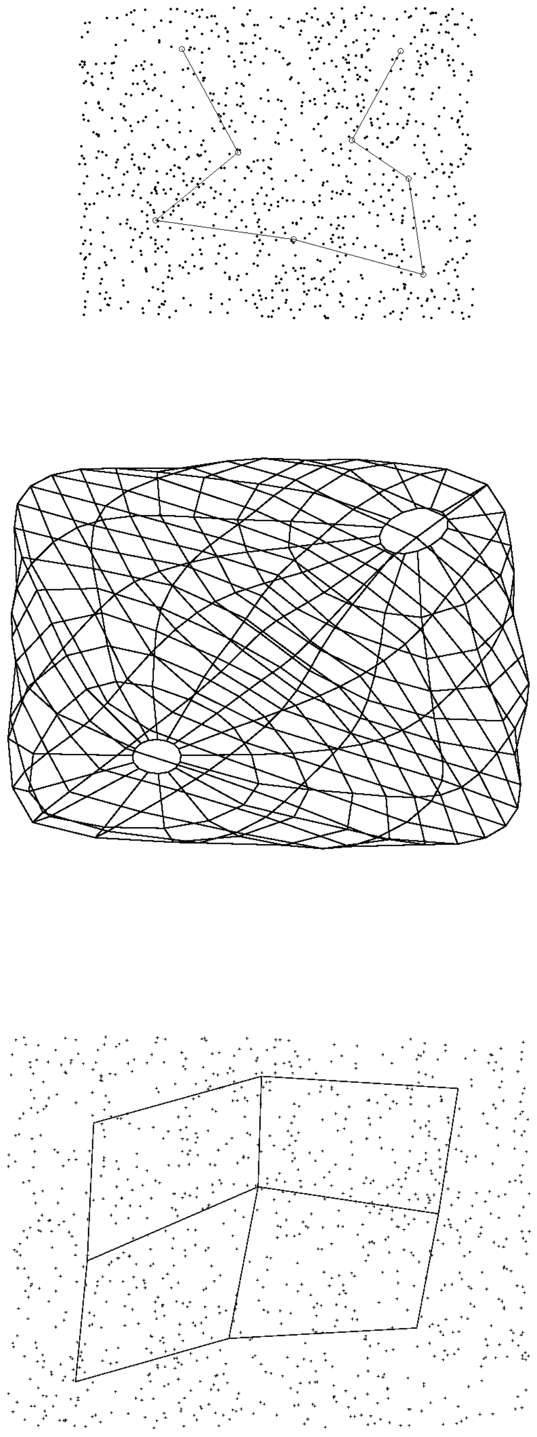

Figure 5: Which mapping is topology preserving? Figure 1 shows a mapping that is not topology preserving, since neighboring cells are mapped to non-neighboring points in $\mathbb{R}^2$. On the other hand, the middle clustering is not topology preserving, because neighboring points in $\mathbb{R}^2$ are mapped to non-neighboring cells in the topology. Only the third picture shows a topology preserving clustering.

Suppose the set of centers is one dimensional:

$$C = 0.1, 0.2, 0.4, 0.5.$$

If a data point such as $x = 0.25$ is chosen, then the winning center index is $w = 2$, and the sorted set of indices would be $V = \{2, 1, 3, 4\}$.

Looking at distances, we see that

$$d_{ng}(1,2) = 1, \quad d_{ng}(2,2) = 0, \quad d_{ng}(3,2) = 2, \quad d_{ng}(4,2) = 3.$$

Connection Update:

The Neural Gas algorithm will also be constructing the connection matrix. The main idea is the following:

> "Let $c^{(w)}$ be the center closest to data point $\boldsymbol{x}$, and let $c^{(k)}$ be its closest center. Then set $M_{w,k} = 1$."

As centers get updated, the closest neighbors will change, so we'll keep track of the **age of the connections** and **remove them** if the have aged past a preset criteria, T^M. That is, we will construct a time matrix T where T_{ij} =age since $M_{i,j}$ was last updated.

As a side remark, we might notice that a connection matrix can be constructed independently of the cluster center updates, so this could be implemented in k-means or any clustering algorithm.

Now for the main algorithm:

The Neural Gas Algorithm:

1. Initialize the centers, M, T, $\epsilon_{i,f}$, $\lambda_{i,f}$, $T^M_{i,f}$ `tmax` (max number of iterations for NG). For example, values that seem to work well are the following (where N=number of data points).

    - $\epsilon_i = 0.3$, $\epsilon_f = 0.05$,
    - $\lambda_i = 0.2N$, $\lambda_f = 0.01$,
    - $T^M_i = 0.1N$, $T^M_f = 2N$
    - `tmax`$= 200N$.

    The parameters with the i, f subscript are updated using Equation (1) described in the last section.

2. Select a data point $\boldsymbol{x}$ at random, and find the winner, $c^{(w)}$.

3. Compute V.

4. Update all centers:

$$c^{(i)} = c^{(i)} + \epsilon \exp\left(\frac{-d_{ng}(i,w)}{\lambda}\right)(\boldsymbol{x} - c^{(i)})$$

5. Update the Connection and Time Matrices:
   - Let $i = V(2)$.
   - If $M_{w,i} = 0$, set $M_{w,i} = 1$, and start the clock $T_{w,i} = 0$.
   - If $M_{w,i} = 1$, just reset the clock $T_{w,i} = 0$.
   - Age all connections by 1, $T_{j,k} = T_{j,k} + 1$.

6. Remove old connections:

   Set $M_{w,j} = 0$ whose corresponding entries in T are larger than the current value of T^M. Note that we set this initially to be small, then larger as training progresses.

7. Repeat.

Notes about results

The Neural Gas algorithm produces what is called an *induced Delaunay Triangulation*, which is what we produced in the introduction of this chapter when we were building a Voronoi Diagram. The induced triangulation will be a subgraph of the full triangulation. See [**?**] for more details on the definitions.

Matlab and Neural Gas

Matlab has not yet implemented a version of the Neural Gas algorithm. We will construct a suite of programs below. Although its not necessary, we will use data structures so that we're familiar with their useage when we get to the neural networks section.

The suite of programs that we'll be using are below. They include: `NeuralGas.m`, `initng.m`, `paramUpdate.m`, and `plotng.m`.

Here are the details (the code will be on our class website).

- The program `initng` is a program that will set all of the training parameters, and initialize the centers. This program should be edited at the beginning of a problem. The code is given below. Note the presence of the data structure, which allows us to wrap the parameters and the data together.

```
function C=initng(X)
[m,n]=size(X);
C.NC=500;            %Number of clusters
C.lr=[0.3 0.05];    %initial, final learning rate epsilon
```

```
C.nbr= [0.2*n 0.01]; %initial, final lambda (neighborhood size)
C.tcon=[0.1*n 2*n];  %initial, final time  (for connection matrix)
C.tmax=200*n;        %max number of iterations
C.epochs=1;          %number of epochs (each epoch runs tmax iterations,
                       %and resets the training parameters after each.)
C.tflag=1            %Training flag: 1=Use Connection, 0=Do not use
Id=randr(n);
C.cen= X(:,Id(1:C.NC)); %Initialize the centers randomly from the data
C.M=zeros(C.NC,C.NC);   %Initialize the connection matrix (if tflag=1)
```

- The Neural Gas main algorithm is presented below. To get things moving a little faster, one can change several things. For example, if we have a large number of centers, we probably don't need to update all of them.

```
function C=NeuralGas(X,C)
%FUNCTION C=NeuralGas(X,C)
%   This is the Neural Gas clustering algorithm.  There are
%   two ways to call this program:
%     [C,M]=NeuralGas(X);
%        With only one input argument, the program will
%        read the file initng.m for the initialization
%        procedures, and train the network.  See the bottom
%        of this file for a sample initng.m file.
%     [C,M]=NeuralGas(X,C)
%        With two input parameters, we assume the centers
%        have been initialized in the structure C.  See the
%        initng file for structure specifications.

%We will use the following as the current update parameters:
%  lr = current learning rate
%  nbr= current neighborhood size (lambda)
%  t  = current time setting (for connection matrix)

%INITIALIZATION:

if nargin==1  %use the initng file
C=initng(X);
end

[m,n]=size(X)  %Dimension is m, number of points is n
[m,numcenters]=size(C.cen)

lr=C.lr(1);
nbr=C.nbr(1);
if C.tflag
   t=C.tcon(1);
   Age=zeros(numcenters,numcenters);
end

for j=1:C.epochs
    for k=1:C.tmax
```

```
if rem(k,1000)==0
  disp('iterate =');
  disp(k)
  disp('out of')
  disp(C.tmax)
end
%*************************************************
%
%   Step 1:  Choose a data point at random and compute
%              distance vector.
%
%*************************************************

curidx=ceil(n*rand);
Nx=X(:,curidx);
D=(C.cen - repmat(Nx,1,numcenters)).^2;
dd=sum(D);
[Md,w]=min(dd);

D=(C.cen - repmat(C.cen(:,w),1,numcenters)).^2;
dd=sum(D);
[Md,Id]=sort(dd);

%*****************************************************
%
%  Step 2:  Update all centers and training parameters
%
%******************************************************

for s=1:numcenters
aa=lr*exp(-(s-1)/nbr);
C.cen(:,Id(s))=C.cen(:,Id(s))+aa*(X(:,curidx)-C.cen(:,Id(s)));
end

lr=paramUpdate(C.lr,k,C.tmax);
nbr=paramUpdate(C.nbr,k,C.tmax);

%*********************************************************
%
%  Step 3:  If necessary, update Connection matrix and Time.
%
%*********************************************************
if C.tflag
   ab=Id(2);
   C.M(w,ab)=1;
   Age(w,ab)=0;
   Ix=find(C.M>0);
   Age(Ix)=Age(Ix)+1;
   Iy=find(Age>=t);
   C.M(Iy)=0;
   t=paramUpdate(C.tcon,k,C.tmax);
end  %End of time and connection update

    end %End of C.tmax (k) loop
end %End of C.epochs (j) loop
```

- The function `paramUpdate` is simply a call to Equation (1).
- The function `plotng` is an example of how we can plot the edges of the graph that is produced using the algorithm. As presented, it only plots the two dimensional graph.

```
function cidx=plotng(C)
%Plots the connections in C as line segments.
%  C is the center structure that is constructed
%    by the neural gas routine.  cflag returns 1
%    if there are no connections.

[m,n]=size(C.M);  %A square matrix
cidx=0;

for i=1:n
  for j=1:n
    if C.M(i,j)==1
      cidx=cidx+1;
      Lx(:,cidx)=C.cen(:,i);
      Ly(:,cidx)=C.cen(:,j);
    end
  end
end

if cidx==0
   disp('No connections');
end

for i=1:cidx
  line([Lx(1,i) Ly(1,i)],[Lx(2,i), Ly(2,i)]);
  if i==1
    hold on
  end
end
```