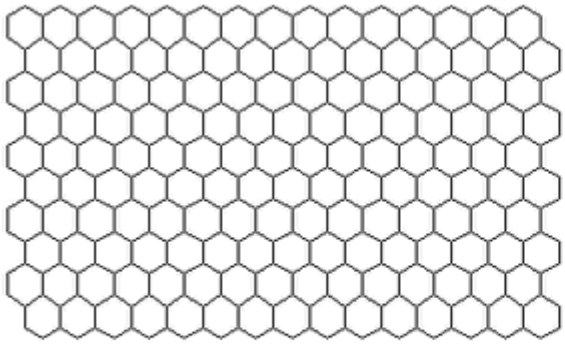
Self organizing networks

The self-organizing feature map, SOFM or SOM, is a neural network tool developed by Kohonen (1984,1995). SOMs are principally used to facilitate the visualization and interpretation of high-dimensional datasets, although they may be applied to address a number of other problems in spatial analysis. In broad terms the SOM approach attempts to find a set of attribute vectors (output vectors, effectively multi-dimensional cluster centers), which are used to represent a large set input attribute vectors. However, unlike most other clustering techniques, the set of output vectors are constrained by the SOM procedure to have a well-defined 2D (map) spatial relationship with one another. The enforced spatial relationship is designed to ensure that similar input vectors are assigned to output vectors that are spatially near to one another in the output map. Step-by-step worked examples of the procedure are provided in Section 8.3.3.2 and Section 8.3.3.3. Software to support the SOM is available in several GIS and remote sensing packages — e.g. Idrisi, ENVI, TNTMips — in all these cases primarily as a form of unsupervised map classification. SOM facilities are also provided in some commercial neural network software packages (e.g. the Neural Network toolbox for MATLab), and the (now defunct) SOM Toolbox for MATLab. The latter was available free of charge under the GNU General Public License from the Helsinki University of Technology, where it was developed. In this subsection we draw on the documentation associated with the SOM toolbox for our description of the procedures involved.

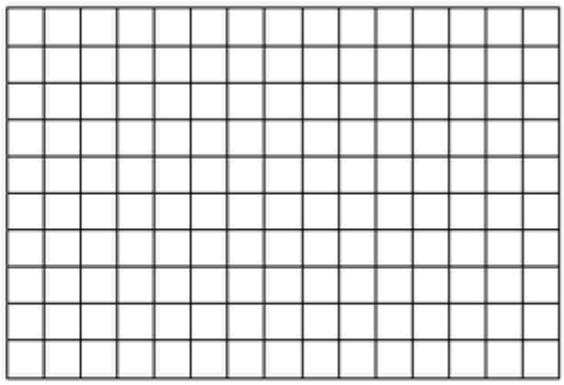
The SOM map itself (the output space) is formed as a regular grid of cells or units (neurons), for example a 10x8 grid (n=80 units) or a 16x32 grid (n=512 units) — in fact the grid can be much larger if desired. The choice of grid size, form and topology may be user-definable or pre-specified, depending on the software tools being utilized. As with other forms of neural network modeling, each grid cell is connected to adjacent grid cells (neurons). However, in SOMs these connections are structured according to a spatial neighborhood relation, and this provides the key distinction between SOM and other forms of multi-dimensional analysis of this type. *Grid size* refers to the number of units or cells to be used, typically arranged as an approximately equal number of rows and columns. *Grid form* refers to the use of square or hexagonal cells (see Figure 8-28) — hexagonal cells may provide a preferable cell neighborhood arrangement. *Grid topology* refers to the way in which the grid boundary is handled. Standard topologies supported are sheet (planar grid, hence 4 edges), cylinder (grid with two opposite edges attached when computing neighborhoods — 2 edges), and toroidal (grid with both pairs of opposite edges attached when computing neighborhoods — no edges).

Each grid cell unit has an associated vector, **m**, that provides a model of the k-dimensional input dataset, hence the n units seek to provide a model representation of the observations (e.g. multi-band source image pixels, attributes of statistical regions in a country), each of which consists of data across k-dimensions (e.g. feature attributes, spectral bands). These model vectors are also known as *prototype vectors* or *codebook vectors*. They can be regarded as a set of points with coordinates reflecting those of the input space or data space. In 1-, 2- or 3-dimensional space the input space and model vectors can be visualized graphically, but with higher dimensions such visualization is only possible by selecting subsets of the dimensions and plotting these individually.

Figure 8-28 SOM grids



Hexagonal SOM grid



Rectangular SOM grid

The SOM performs two functions: (i) the modeling of the source data with a smaller number of vectors that attempt to represent the source data as closely as possible (e.g. minimizing some measure of deviation from the observations); and (ii) the production of a topology in the SOM grid whereby similar models are close together and dissimilar models are far apart. The resulting SOM grid is not a geographic map, but a two-dimensional representation or projection of the similarity of the models applied. Typically the SOM is displayed as a gray scale or in some cases a simple colored grid (rectangular or hexagonal lattice) with light gray or similar colors indicating clusters of similar data, whilst dark or dissimilar colors indicate divisions in the underlying models (in some implementations this representation is inverted, so that darker shades indicate similar classes). The SOM is thus a form of *data mining* tool, which may be used for this purpose alone, or as a precursor to further analysis and data classification (with cells prototype vectors providing the classes). If the classification is then applied to a source spatial dataset, for example a remotely-sensed image, a map of the countries of the world, a set of point observations (e.g. medical cases and controls), or a network of streets (see Jiang, 2004) the procedure can generate a coded geographic map in addition to its functional map. Other forms of non-geographic visualization are also available, including analysis of the individual SOM unit vectors. Currently SOM procedures are provided as tools for multi-band image classification within a number of software packages, including Idrisi and TNTMips. In the following subsections we provide an example and discussion based on the TNTMips SOM implementation and we take a brief look at the Idrisi SOM facility. A second example is then examined where the classic traveling salesman problem (TSP) is addressed using SOM concepts.

SOM unsupervised classification of hyper-spectral image data

A neural network-based unsupervised classification procedure known as a Self-Organizing Map (SOM) can be developed from an extension to the single spectrum angular mapping (SAM) analysis described in Section 4.2.12, Classification and clustering. In the following example the same 512x614 pixel image with 224 spectral layers provides the sample dataset. The procedure operates broadly as per the earlier discussion. The number of classes in this process is preset, in this case to 512, using a 16x32 SOM neural network. Each cell in the net, 1…*k*, is initially assigned a random vector, $\mathbf{m}_k$, of values — typically these will be random numbers for each spectral band drawn from a uniform distribution over the observed range for the band in question or for all bands in the image.

The next step is to train the neural network by comparing a randomly selected pixel from the source image that has spectral vector **x** say, to the vectors of each neural network unit. This comparison is achieved using a *norm*:

$$d_{xk} = \|\mathbf{x} - \mathbf{m}_k\|$$

that measures the separation between the sample spectral vector, **x**, and the k^{th} unit in the neural network, $\mathbf{m}_k$. The measure most commonly applied is of the form:

$$d_{xk} = \sum_i (x_i - m_{ki})^2$$

where x_i and m_{ki} are the i^{th} elements of their respective vectors. The *best matching unit* (BMU) is the SOM unit, *c* say, which has the smallest separation from the input vector as computed using the selected norm. The components or weights of this BMU vector are then modified using a very simple general formula:

$$m_{ci}(t+1) = m_{ci}(t) + p(t)\left[x(t) - m_{ci}(t)\right]$$

where *t* is the time period (iteration number or epoch) and $p(t)$ is an adjustment factor that varies with time, *t*. In practice $p(t)$ consists of two components:

$$p(t) = a(t)h_{cj}(t)$$

The first of these components, $a(t)$, is known as the learning rate, and is simply a fractional value that is (typically) adjusted over time. For example, *if T* is the total number of steps or epochs in the training period, $a(t)$ might be taken as a simple decreasing linear function of time over the interval $[0,T]$, starting at $a(0)=0.5$:

$$a(t) = a(0)(1-t/T) \text{ or } a_t = a_0(1-t/T)$$

The second component is the grid neighborhood time-adjusted kernel function, $h_{cj}(t)$. All vectors (*j*) within a specified neighborhood of the BMU, *c* (typically all those that relate to adjacent grid cells and/or within a specified radius, *r*, of *c*) are adjusted using a pre-selected formula, i.e. not just the central BMU, at *c*. In its simplest form, all map vectors (*j*) within grid radius *r* of *c* are adjusted equally, i.e. $h_{cj}=1$ if distance

$$d_{cj} \leq r$$

otherwise $h_{cj}=0$. Two-dimensional kernel functions of the type previously described in connection with density modeling are commonly used (see further, Table 8-7), particularly Gaussian or truncated Gaussian functions. This adjustment can be seen as migrating point *c* in *k*-space and its near neighbors towards the sample point **x**. This process, which is a form of sequential regression, is repeated for a substantial sample of pixels often involving thousands of iterations. If the input data are 1-, 2- or 3-dimensional the progress of the adjustment process can be visualized, for example by displaying the results every 10 epochs, providing a form of video of the progress of the procedure.

When the process has completed the individual units or grid cells will have converged to a set of vectors that provide a form of 'best representation' of the sample data. The SOM grid will contain similar vectors in close proximity to each other. These vectors can then be used to create a classified version of the source image, using the spectral angle method described earlier. The spectrum of each pixel in the source dataset is compared with the model spectra in the SOM grid and the BMU determines the classification applied. Coding nearby units in the SOM with similar or the same colors and applying these to the source dataset produces a geographic map that both identifies regions that are similar in spectral data and at the same time offers a fine level of discrimination. Figure 8-29 illustrates this classification process for the Cuprite dataset discussed earlier in this Guide (Section 4.2.12, Classification and clustering).

Figure 8-29 SOM classification of remotely-sensed hyperspectral data

Issues: There are a number of important observations that should be made regarding the SOM procedure described:

- **Initialization**: the choice of the initial set of map vectors can influence both the speed of the computation and its outcome. Speed may be considerably improved by judicious choice of initial vectors in some problems, i.e. rather than random vector assignment. Multiple runs of the procedure with different initialized vectors may help to produce more consistent or robust results than a single run. For example, it is entirely possibly that two or more groups of similar vectors will be located within the SOM, but which remain isolated from each other rather than forming a single, cohesive grouping
- **Pre-processing**: with remote sensed data, factors such as atmospheric scattering, topography, surface lighting (solar effects) and data capture variables (e.g. view angle, stripes) may warrant pre-processing of the data to minimize or remove known effects
- **Normalization**: whilst it may be reasonable to expect the data values of a hyperspectral image to have a broadly similar range across the spectral bands this is not the case for general attribute data. For example, with attributes such as cases of a particular disease in sampled zones the data range might be 0-100, whilst other variables, such as the zone population or a pollutant measurement, may be in a much larger range. The latter variable might dominate the BMU computation and hence hide variations in key attributes. Pre-normalization of the data vectors, for example by ensuring each has unit variance, may be advisable in such cases. Normalizations supported within the SOM Toolbox include: variance (vectors normalized to unit variance and 0 mean); range (vector components normalized to a [0,1] range); log (simple logarithmic transformation); softmax (vector components are initially variance normalized and the results converted using the logistic function into a [0,1] range with logistic curve form); and histogram (data values are replaced by histogram bins with approximately equal numbers of observations in each bin. Bins are ordered 1, 2, 3 etc. and then finally scaled to give a [0,1] data range. Note that output data and visualizations can be produced for non-normalized, normalized or for data that has been normalized and then de-normalized prior to output
- **Missing data**: the computation of the distance metric assumes that all spectral bands or all attribute sets in the sample and final datasets are present. If one or more gaps in the data exist this can either be ignored and assumed to have little or no impact on the result, or the missing attribute data could be assumed to apply to all samples, hence effectively removing this component from the analysis
- **Masking and weighting**: it may be desirable to mask off (binary weight) or apply a variable weighting to some data in the sample or final datasets. This could be at the attribute level or the feature level
- **Learning and tuning**: it is commonly the case that different rates of adjustment are required for an initial learning period (sometimes called the ordering phase) and the subsequent fine-tuning phase of learning. In both instances similar time-dependent functions may be used, but each with different initial parameters. For example, the default parameters in the MATLab Neural Network toolbox are: 0.9 (ordering-phase learning rate, for an initial 1000 steps); max grid cell separation distance (ordering-

phase neighborhood distance>1); 0.02 (tuning-phase learning rate, for subsequent steps); and 1 (tuning-phase neighborhood distance, i.e. immediate neighbors only)

- **Distance metrics**: a variety of grid or lattice distance metrics, d_{cj}, may be employed. Examples include: Euclidean; Manhattan (rectilinear); Box distance (directly adjacent cells in a rectangular grid 3x3 cell Moore neighborhood=1 distance unit away, or the outer cells of an expanded Moore neighborhood of 5x5 cells=2 units away etc.); and Link distance (based on connectivity steps between grid cells or nodes). Note that calculations, especially at grid edges, will vary according to the chosen grid form and topology (e.g. rectangular or hexagonal; sheet, cylinder or toroidal)
- **Neighborhood functions and Learning rate functions**: Table 8-7 provides a summary of some of the commonly used functions employed. Essentially the neighborhood functions are kernel functions of the type encountered previously

Table 8-7 SOM neighborhood and learning rate functions

Functions	Name	Expressions
Neighborhood functions	Bubble	$h_{cj}(t) = \max\left[0, 1 - \left(\sigma_t - d_{cj}\right)\right]$
	Gaussian	$h_{cj}(t) = \exp(-d_{cj}^2 / 2\sigma_j^2)$
	Cut Gaussian	$h_{cj}(t) = \exp(-d_{cj}^2 / 2\sigma_j^2) 1\left(\sigma_t - d_{cj}\right)$
	Epanechnikov	$h_{cj}(t) = \max\left[0, 1 - \left(\sigma_t - d_{cj}\right)^2\right]$
where σ_t is the neighborhood radius at time t, d_{cj} is the distance between map units c and j on the grid using the chosen grid metric, and 1(x) is the step function 1(x)=0 if x<0 and 1(x)=1 if x>=0		
Learning rate functions	Linear	$\alpha(t) = \alpha_0 (1 - t / T)$
	Power	$\alpha(t) = \alpha_0 (0.005 / \alpha_0)^{t/T}$
	Inverse	$\alpha(t) = \alpha_0 (1 + 100t / T)$
where T is the overall training length (steps) and α_0 is the initial learning rate		

Idrisi (V15 and later) includes a number of neural network modeling tools, principally provided to facilitate image analysis and classification. In Section 8.3.1, Introduction to artificial neural networks, we showed an example of the use of Idrisi's MLP neural network facility as part of a landcover change modeling (LCM) process. In Figure 8-30 and Figure 8-31 we show the standard Idrisi SOM NN facility applied to a multi-band satellite image set. We have selected unsupervised training, but supervised training is also supported by this module. As with the MLP module, multiple input rasters are specified — here they consist of 3 SPOT images covering a region of Malawi in East Africa, with one of the bands de-striped before processing (Band 2). The input layer is thus 3 nodes, whilst the SOM output layer must be specified as an *n*-by-*n* grid. We have specified an 8x8 grid with initial clustering performed by *k*-means and a maximum of 10 clusters. The default initial neighborhood radius was selected which reduces to 1 as training proceeds. The process commences with the 'coarse tuning' phase, during which the network attempts to identify clusters in the input data and group similar clusters in the output layer. The second stage is to classify the map using the learned weights, assigning a cluster ID to every pixel of the image space. In this case nine clusters were identified and then mapped as shown in Figure 8-31.

Figure 8-30 Self Organizing Map (SOM) classification — Idrisi

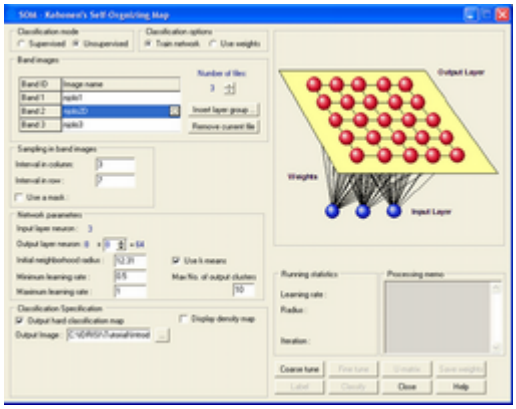
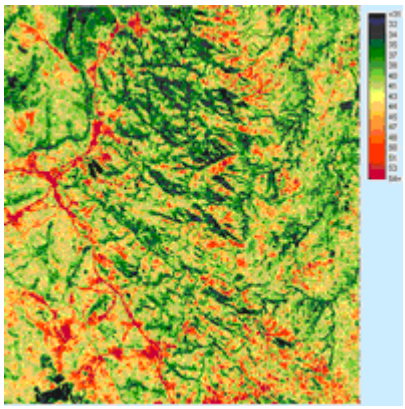
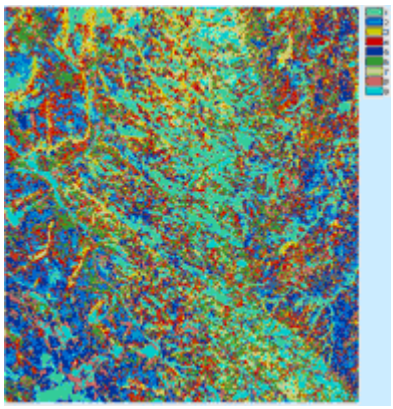


Figure 8-31 SOM classified 3-band image

A. Source image — SPOT band 1	B. SOM classified result (9 classes)
	

TSP optimization using SOM concepts

If the input dataset is comprised of *m* points in 2 dimensions, the coordinates can be viewed as *m* sets of 2-element vectors. The SOM procedure can be run with *n*<*m* initial units which could be initially set to random coordinates, all set to the same coordinate (e.g. the median of the *x*- and *y*- ranges in the source data), or a simple linear spread across one or both axes. As training proceeds, the initial positions of the model vectors (i.e. the locations of the units in 2-space) will migrate towards the source data points in a manner which will minimize a separation measure of the form:

$$d_{xm} = \sum_i (x_i - m_i)^2$$

This process is essentially one of minimizing the sum of squares, so as noted earlier is a form of sequential regression. But the SOM not only adjusts the BMU but also its neighbors. This observation has led some authors to consider a variation on the SOM method that can be used as an heuristic to solve the travelling salesman problem (TSP). The algorithm we describe is due to Angeniol *et al.* (1988), but it has since been improved by other authors and applied to well-known test problems (such as TSPLIB test ATT532). The idea with this approach is to create a tour of *M* cities in the plane by commencing with a set of units or nodes connected together to form a 1-dimensional ring, rather like an elastic band. The objective is to progressively move the nodes that form this ring towards the *M* cities through an iterative process, with an actual tour being obtained when each city has "captured" one node of the ring. The algorithm is as follows:

- Cities are numbered *i*=1 to *M* and assigned coordinates:

  $\left(x_1^i, x_2^i\right)$

- Nodes of the ring are assigned coordinates:

$\left(c_1^j, c_2^j\right)$

- Initially a single node (N=1) is generated as (0,0). Nodes are added and deleted as the process proceeds. Each node on the ring has two neighbors, these being nodes j-1 (mod N) and j+1 (mod N)
- Select each city, i, in turn (sequentially, from 1 to M) in a pre-defined random order that is kept constant throughout the processing. Find the node c on the ring that is closest to i (minimum squared Euclidean distance). Move node c and its ring neighbors closer to i according to the rules described below and continue
- If the node c is the closest node to more than one city, create a new node with the same coordinates. This will clearly occur frequently in the early stages of iteration, and the number of nodes will grow to at most 2M (from experimental evidence)
- If a node has not been selected as the closest to a city for 3 iterations it is deleted
- The movement rule applied is:

$$c_k^j \leftarrow c_k^j + f(G,n)\cdot(x_k^i - c_k^j)$$

- which can be compared with the basic SOM rule:

$$m_i^{t+1} \leftarrow m_i^t + p^t \cdot \left(x^t - m_i^t\right)$$

- In this case the function $f(G,n)$ is of Gaussian form with n being the distance around the ring of each node from that chosen (where this is computed as the least distance) and G is a parameter that the authors describe as the gain. The function is of the form:

$$f(G,n) = \frac{1}{\sqrt{2}} e^{-(n/G)^2}$$

- The parameter G is reduced during each completed processing of the M cities by a factor (1-a) where 0<a<1 (e.g. a=0.2). When G is large all nodes move towards city i with the same strength, whereas as G tends to 0 only the closest node moves towards city i. Simulation tests on sets of up to 1000 cities showed good results (i.e. within a small percentage of the known optimum value) within modest computational time — decreasing the parameter a produces better results but increases the processing time considerably. A more recent test using a slightly modified version of this algorithm applied to the ATT532 test dataset (532 US cities) produced a best result of 28,658 as compared with the known optimum solution of 27,686 (this can be verified using the Concorde software package and selecting the ATT metric — this metric is simply a Euclidean metric with an adjustment factor of the square root of 0.1)