Self-organizing

Antanas Verikas
antanas.verikas@hh.se

IDE, Halmstad University

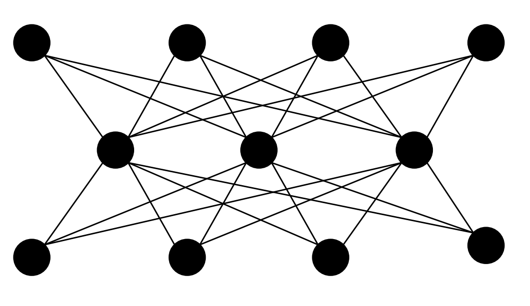
2013

Definitions

- No supervisor available.
- Try to find some order in the environment.
- Clusters (i.e.data is not homogeneously distributed)
- Directions (i.e. some projections carry more information than others)

One hidden layer MLP

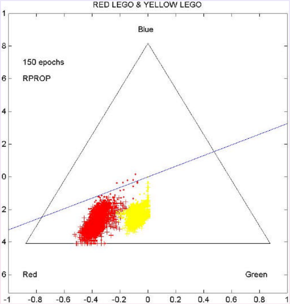
- Output = Input



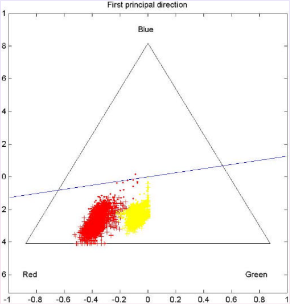
- Input

- The auto-encoder is trained to reproduce the output through a "bottle neck" - must try to find an efficient coding.
- Will lead to $\sim$ principal components.

MLP auto-encoder example

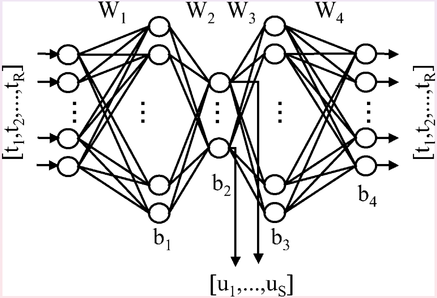


- 2-1-2 MLP with linear output but nonlinear hidden, trained to reproduce the input data.
- The direction of the weight vector **w** for the hidden unit.



- The first eigenvector for the Lego data ("red" and "yellow") covariance matrix.
- The line shows the direction of the first eigenvector.

AANN with orthogonal bottleneck layer outputs



Objective function F

$$F = \eta \mathrm{SSE} + (1-\eta)\Phi$$

$$\Phi = \sum_{i=1}^{S} \sum_{j=i+1}^{S} \frac{s_{ij}}{\sqrt{s_i s_j}}$$

s_i, s_j – variance of u_i and u_j

s_{ij} – covariance of u_i and u_j

Objective function

For K cluster centers $\mathbf{w}_k$ minimize:

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} \lambda_k[\mathbf{x}(n)] \|\mathbf{x}(n) - \mathbf{w}_k\|^2$$

where

$$\lambda_k[\mathbf{x}(n)] = \begin{cases} 1, & \text{if } \mathbf{w}_k \texttt{ is closest to } \mathbf{x}(n) \\ 0, & \text{otherwise} \end{cases}$$

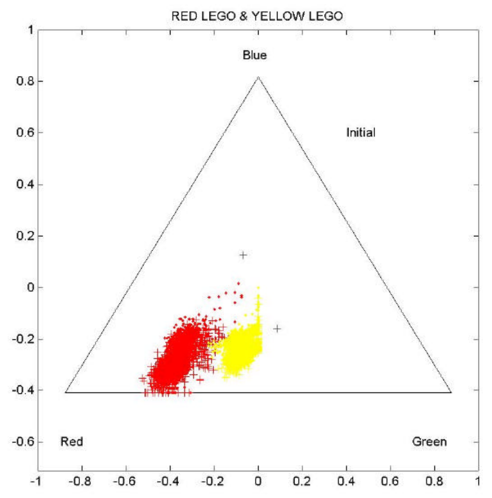
k-means update

k-means can be done in batch or on-line mode:

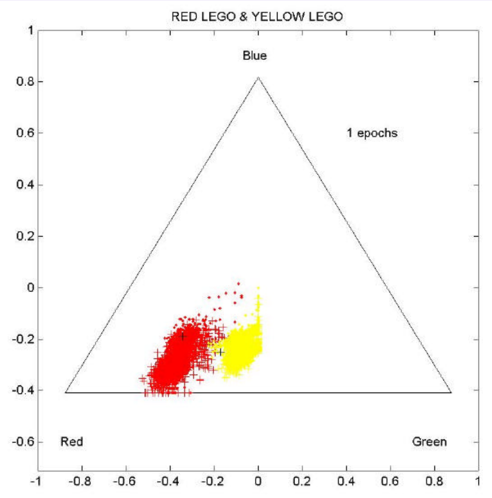
$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \eta \sum_{n=1}^{N} \lambda_k[\mathbf{x}(n)][\mathbf{x}(n) - \mathbf{w}_k]$$

$$\mathbf{w}_k(t+1) = \begin{cases} (1-\eta)\mathbf{w}_k(t) + \eta\mathbf{x}(n), & \text{for closest } \mathbf{w}_k \\ \mathbf{w}_k(t), & \text{otherwise} \end{cases}$$

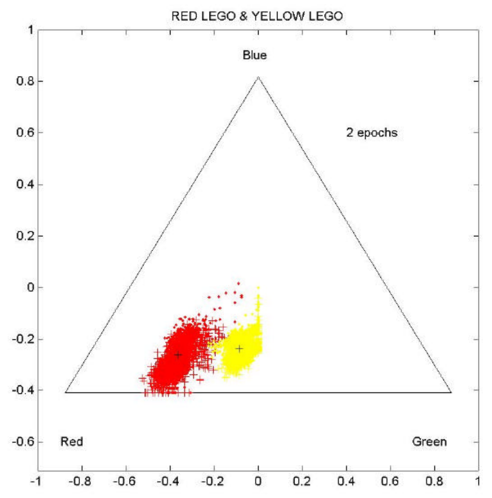
Two initial cluster centers



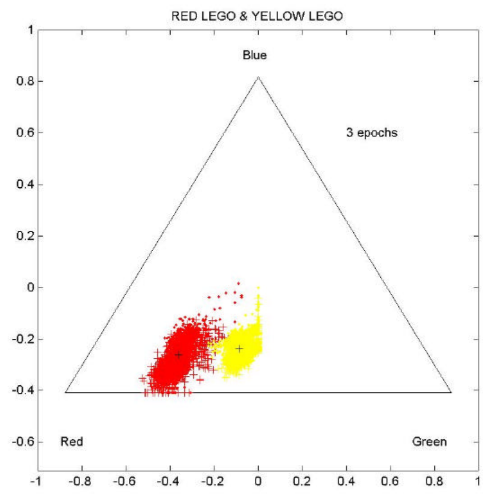
After one epoch



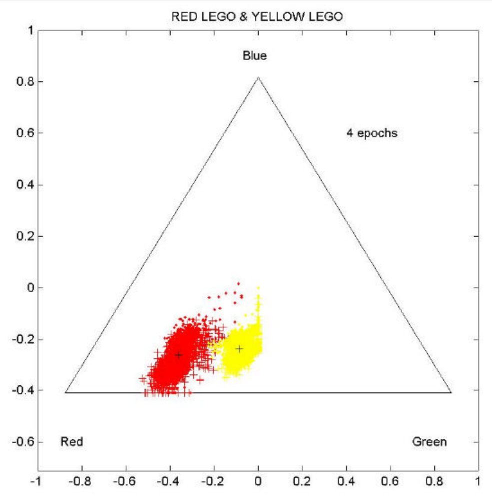
After two epochs



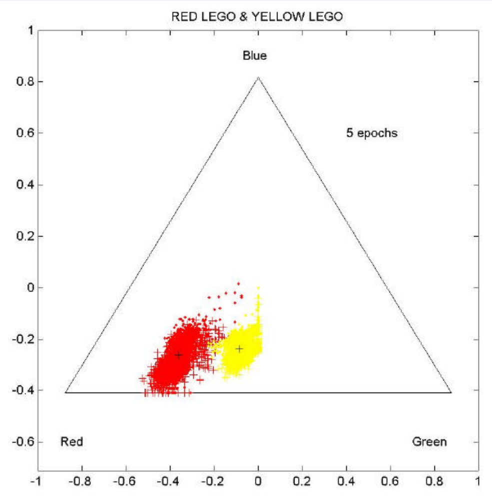
After three epochs



After four epochs

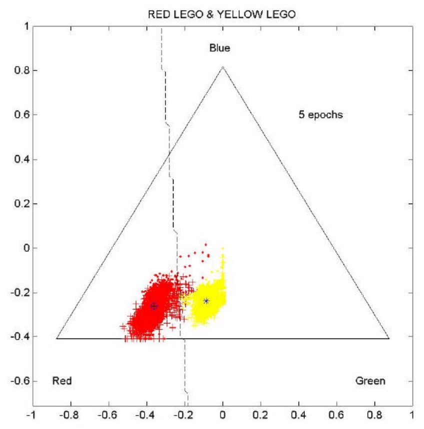


After five epochs

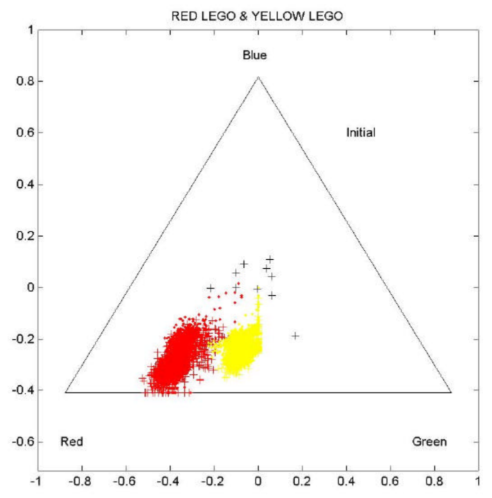


Decision boundary based on the centers

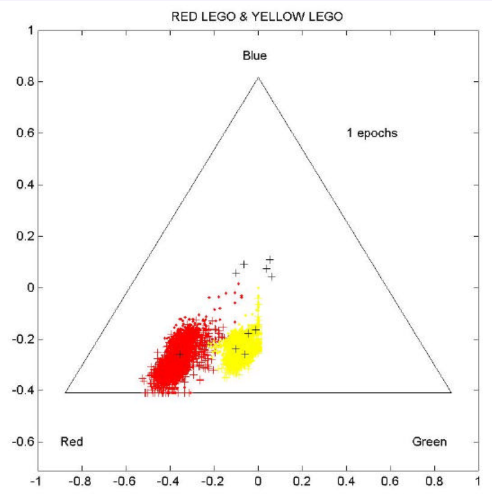
Training error = 0.56%, Test error = 0.80%. The algorithm wasn't told about red & yellow.



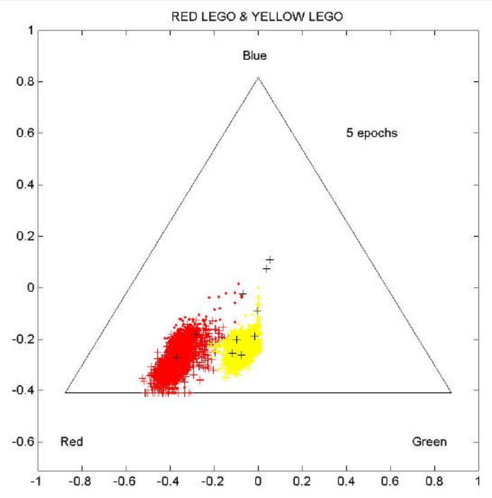
Ten initial cluster centers



After one epoch

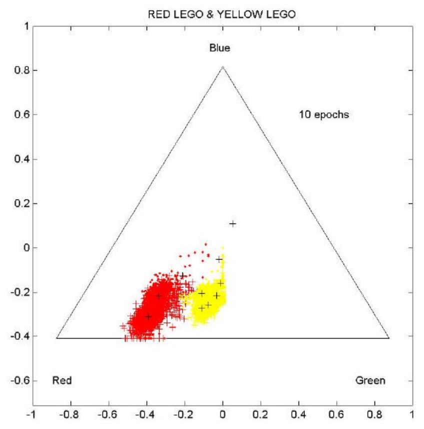


After five epochs



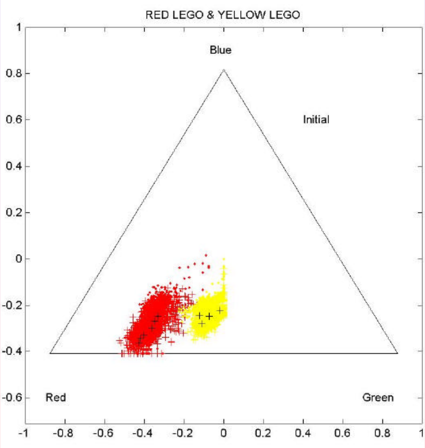
After ten epochs

Takes a long time to get vectors **w** to converge into region of interest.

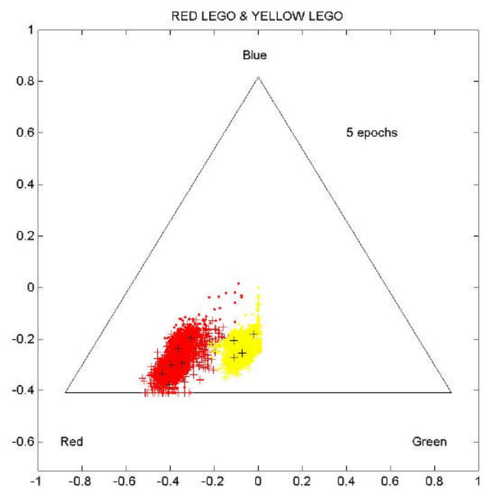


Initial centers from data

"Better" to pick initial points randomly from data.

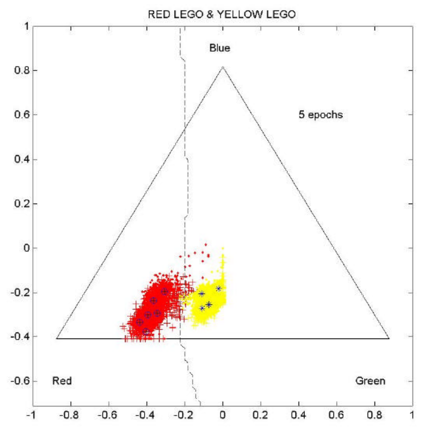


After five epochs



Decision boundary based on the centers

Training error = 0.55%, Test error = 0.52%. The algorithm needs to know about red & yellow.



LVQ

For correctly classified patterns:

$$\mathbf{w}_k(t+1) = \begin{cases} (1-\eta)\mathbf{w}_k(t) + \eta\mathbf{x}(n), & \text{for closest } \mathbf{w}_k \\ \mathbf{w}_k(t), & \text{otherwise} \end{cases}$$

For incorrectly classified patterns:

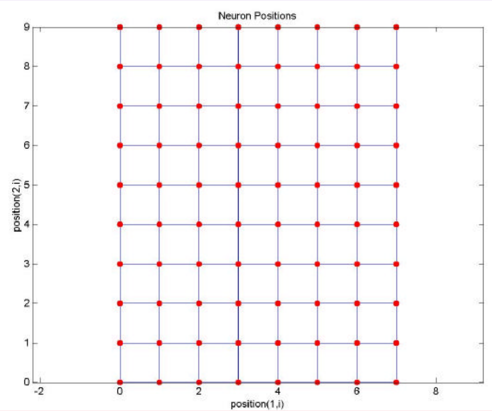
$$\mathbf{w}_k(t+1) = \begin{cases} (1-\eta)\mathbf{w}_k(t) - \eta\mathbf{x}(n), & \text{for closest } \mathbf{w}_k \\ \mathbf{w}_k(t), & \text{otherwise} \end{cases}$$

SOM features

- Impose a topology among the "neurons" (nodes), i.e. define neighborhood relationships.
- Update neighbours along with closest unit.
- Encode the data in a 1D, 2D or 3D sub-manifold.

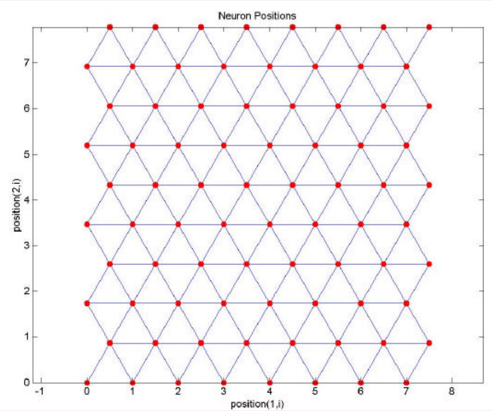
A 2D square lattice topology

Every neuron has 4 near neighbours.



A 2D hexagonal lattice topology

Every neuron has 6 near neighbours.



SOM objective function

For K cluster centers $\mathbf{w}_k$ minimize:

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} \lambda_k[\mathbf{x}(n)] \|\mathbf{x}(n) - \mathbf{w}_k\|^2$$

where

$$\lambda_k[\mathbf{x}(n)] = \begin{cases} 1, & \text{if } \mathbf{w}_k \texttt{ is closest to } \mathbf{x}(n) \\ & \text{or if } \mathbf{w}_k \text{ is neighbour to closest unit} \\ 0, & \text{otherwise} \end{cases}$$

SOM update

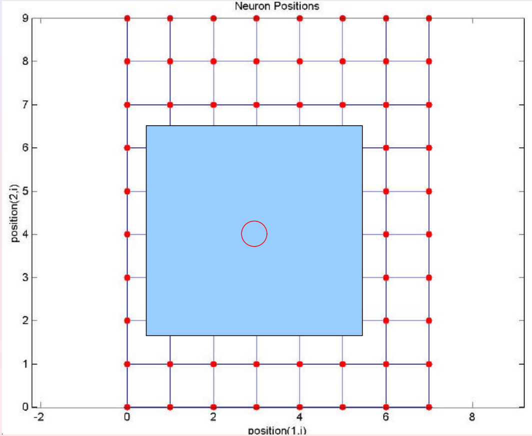
Let node j be the closest node to $\mathbf{x}(n)$.

$$\begin{aligned} \mathbf{w}_k(t+1) &= (1-\eta\lambda_{jk})\mathbf{w}_k(t) + \eta\lambda_{jk}\mathbf{x}(n) \\ \lambda_{jk} &= \exp\left[\frac{-d_{jk}}{2\sigma^2}\right] \end{aligned}$$

where d_{jk} is distance in lattice and σ is decreased with time.

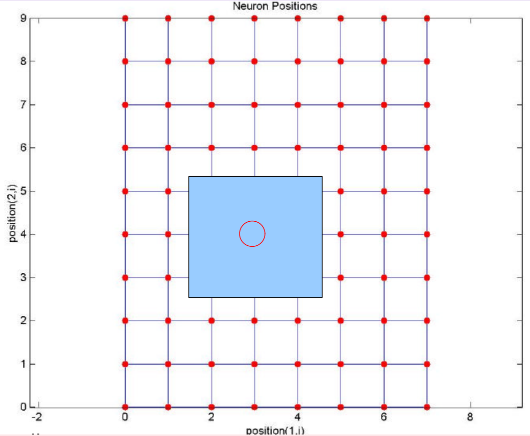
SOM neighbourhood (big)

First, big neighbourhood.



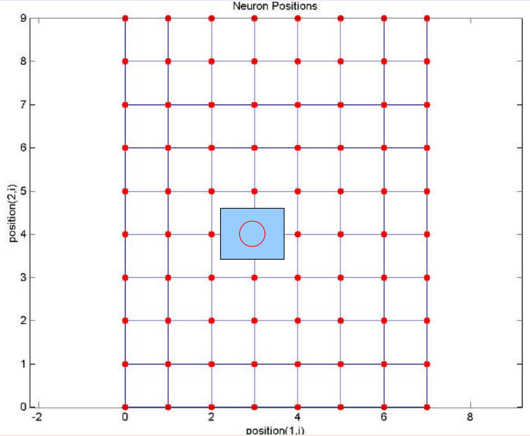
SOM neighbourhood (smaller)

Then, smaller neighbourhood.



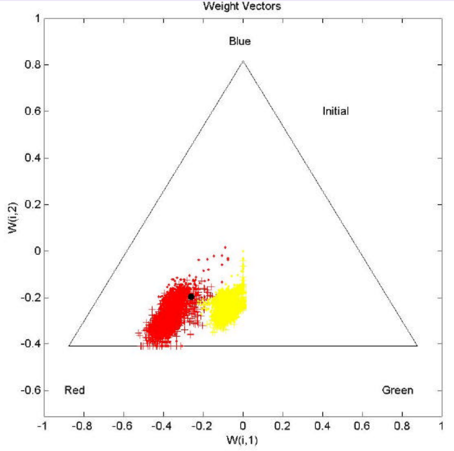
SOM neighbourhood (no)

Then, no neighbourhood.



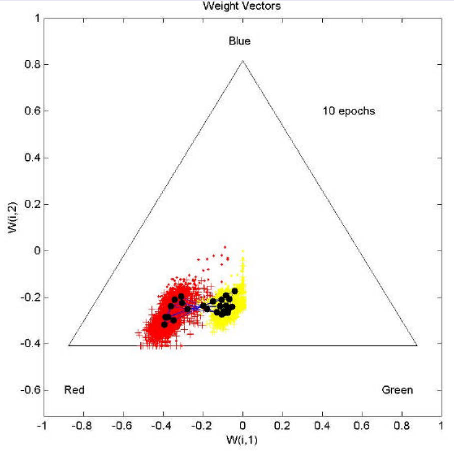
2D map

Initial node positions of 2D map.



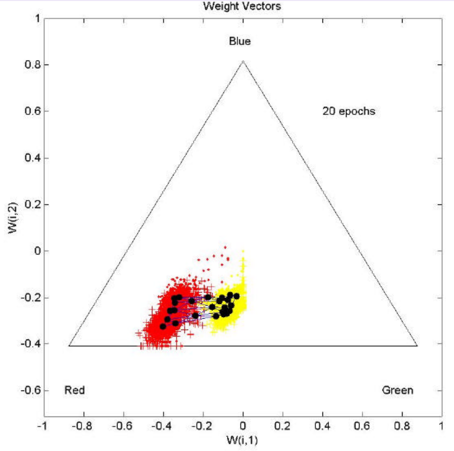
2D map

Node positions after 10 epochs.



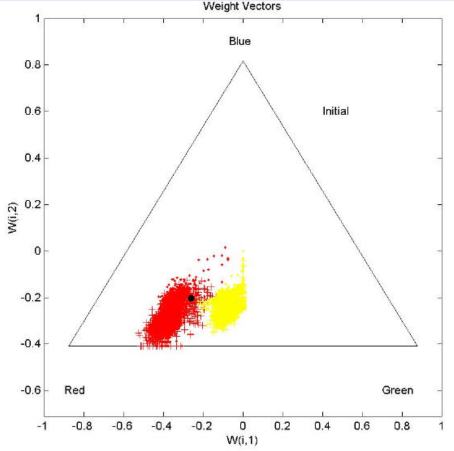
2D map

Node positions after 20 epochs.



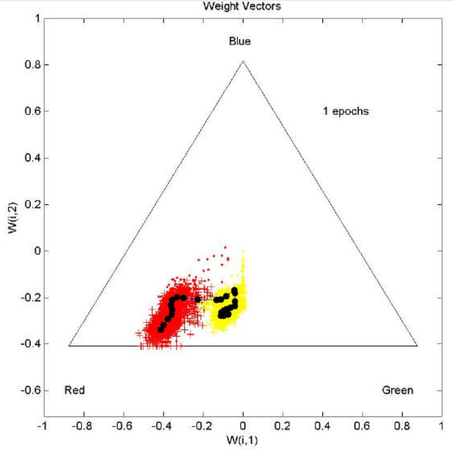
1D map

Initial node positions of 1D map.



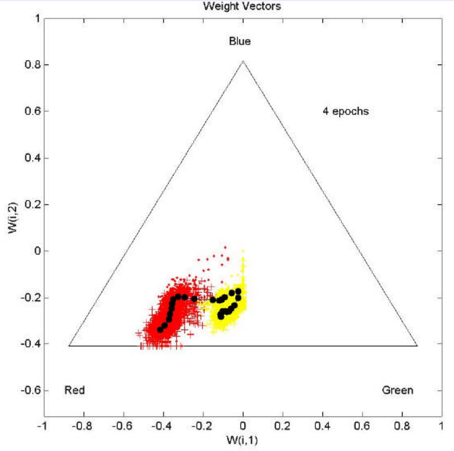
1D map

Node positions after 1 epoch.



1D map

Node positions after 4 epoch.



Other techniques

1. Multidimensional scaling (MDS)
2. t-Stochastic Neighbor Embedding
3. Curvilinear component analysis

Multidimensional scaling

Proximities: π_{ij}; Dissimilarities: $1 - \pi_{ij} = \delta_{ij}$ A solution is obtained by minimizing the *Stress* function:

$$L_1(\mathbf{x}) = \left(\frac{\sum_{i=2}^{M} \sum_{j=1}^{i-1} w_{ij}(\delta_{ij} - d_{ij}(\mathbf{x}))^2}{\sum_{i=2}^{M} \sum_{j=1}^{i-1} w_{ij} d_{ij}^2(\mathbf{x})} \right)^{1/2} \tag{1}$$

where, M is the number of observations, w_{ij} is a user defined weight (for missing dissimilarities w_{ij} is usually set to zero), $d_{ij}(\mathbf{x})$ denotes the Euclidean distance in a q-dimensional space between observations i and j — rows i and j of the $M \times q$ matrix $\mathbf{X}$.