Chapter 12

Towards the Self-Organizing Feature Map

Neural Networks: A Classroom Approach
Satish Kumar
Department of Physics & Computer Science
Dayalbagh Educational Institute (Deemed University)

Properties of Stochastic Data

- Impinging inputs comprise a stream of stochastic vectors that are drawn from a stationary or non-stationary probability distribution
- Characterization of the properties of the input stream is of paramount importance
  - simple average of the input data
  - correlation matrix of the input vector stream

Properties of Stochastic Data

- Stream of stochastic data vectors:
  - Need to have complete information about the population in order to calculate statistical quantities of interest
  - Difficult since the vectors stream is usually drawn from a real-time sampling process in some environment
- Solution:
  - Make do with estimates which should be computed quickly and be accurate such that they converge to the correct values in the long run

Self-Organization

- Focus on the design of *self-organizing systems* that are capable of extracting useful information from the environment
- Primary purpose of self-organization:
  - the discovery of significant patterns or ***invariants*** of the environment without the intervention of a teaching input
- Implementation: Adaptation must be based on information that is available locally to the synapse—from the pre- and postsynaptic neuron signals and activations

Principles of Self-Organization

- Self-organizing systems are based on three principles:
  - Adaptation in synapses is self-reinforcing
  - LTM dynamics are based on competition
  - LTM dynamics involve cooperation as well

Hebbian Learning

- Incorporates both exponential forgetting of past information and asymptotic encoding of the product of the signals

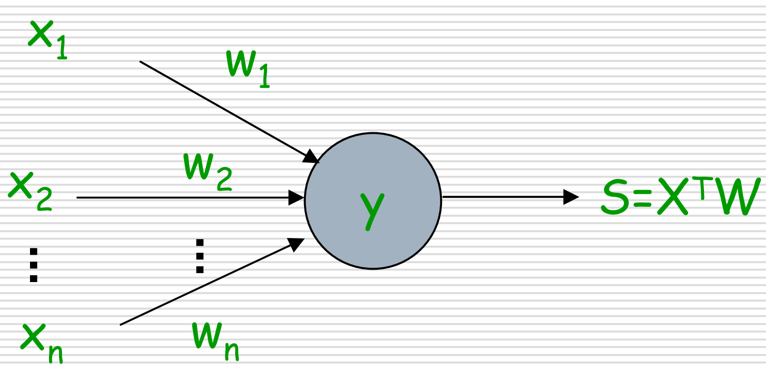
$$\dot{w}_{ij} = -w_{ij} + \mathcal{S}_i(x_i)\mathcal{S}_j(x_j)$$

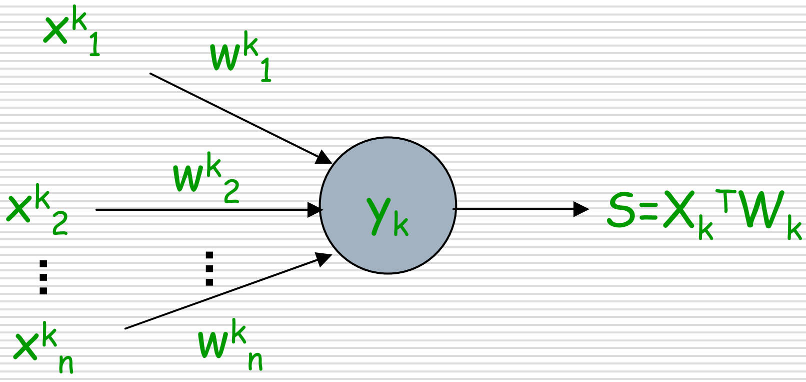
- The change in the weight is dictated by the product of signals of the pre- and postsynaptic neurons

Why is Hebbian Learning Useful?

- A single linear unit using Hebbian learning can extract the dominant eigen-direction of the correlation matrix of an input vector stream that comprises patterns drawn from some unknown probability distribution

Linear Neuron and Discrete Time Formalism





(a) A linear neuron **(b) Discrete time formalism**

Activation and Signal Computation

- Input vector X is assumed to be drawn from a stationary stochastic distribution
- $X = (x_1^k, \ldots, x_n^k)^T$, $W = (w_1^k, \ldots, w_n^k)^T$

$$s_k = y_k = \sum_{i=1}^{n} w_i^k x_i^k = X_k^T W_k$$

$$\begin{aligned} \dot{w}_{ij} &= -w_{ij} + x_i s \\ &= -w_{ij} + x_i y \end{aligned}$$

☜ Continuous

Discrete ☞

$$\begin{aligned} w_i^{k+1} &= w_i^k + \alpha x_i^k s_k \\ &= w_i^k + \alpha x_i^k y_k \end{aligned}$$

Vector Form of Simple Hebbian Learning

- The learning law perturbs the weight vector in the direction of X_k by an amount proportional to
  - the signal, s_k, or
  - the activation y_k (since the signal of the linear neuron is simply its activation)

$$
\begin{aligned}
W_{k+1} &= W_k + \alpha s_k X_k \\
&= W_k + \alpha y_k X_k \\
&= W_k + \alpha_k X_k
\end{aligned}
$$

One can interpret the Hebb learning scheme of as adding the impinging input vector to the weight vector in direct proportion to the similarity between the two

Points worth noting…

- A major problem arises with the magnitude of the weight vector—it grows without bound!
- Patterns continuously perturb the system
- Equilibrium condition of learning is identified by the weight vector remaining within a neighbourhood of an equilibrium weight vector
- The weight vector actually performs a *Brownian motion* about this so-called equilibrium weight vector

Some Algebra

- Re-arrangement of the learning law:

$$\begin{aligned} W_{k+1} &= W_k + \alpha (X_k^T W_k) X_k \\ &= W_k + \alpha X_k (X_k^T W_k) \end{aligned}$$

$$W_{k+1} - W_k = \alpha X_k X_k^T W_k$$

- Taking expectations of both sides

$$\begin{aligned} E[W_{k+1} - W_k] = E[\Delta W_k] &= \alpha E[X_k X_k^T] W_k \\ &= \alpha \mathbf{R} W_k \end{aligned}$$

Equilibrium Condition

- $\hat{W}$ denotes the equilibrium weight vector: the vector towards the neighbourhood of which weight vectors converge after sufficient iterations elapse
- Define the equilibrium condition as one such condition that weight changes must average to zero:

$$E[\Delta W_k] = 0 = \alpha \mathbf{R} \hat{W}$$

$$\mathbf{R}\hat{W} = 0 = \lambda_{\text{null}} \hat{W}$$

- Shows that $\hat{W}$ is an eigenvector of **R** corresponding to the degenerate eigenvalue $\lambda_{null} = 0$

Eigen-decomposition of the Weight Vector

- In general, any weight vector can be expressed in terms of the eigenvectors:

$$
\begin{aligned}
W &= \sum_{i=1}^{m} \beta_i \eta_i + W_{\text{null}} \\
&= \sum_{i=1}^{m} \beta_i \eta_i + \sum_{j=1}^{p} \gamma_j \eta'_j
\end{aligned}
$$

- W_{null} is the component of W in the null subspace, η_i, η_j' are eigenvectors corresponding to non-zero and zero eigenvalues respectively

Average Weight Perturbation

- Consider a small perturbation about the equilibrium:

$$E[\Delta W_k] = \alpha \mathbf{R}(\hat{W} + \epsilon) = \alpha \mathbf{R}\hat{W} + \alpha \mathbf{R}\epsilon = \alpha \mathbf{R}\epsilon$$

- Expressing the perturbation using the eigen-decomposition:

$$\epsilon = \sum_{i=1}^{m} \beta_i \eta_i + \sum_{j=1}^{p} \gamma_j \eta'_j$$

- Substituting back yields:

$$E[\Delta W_k] = \alpha \mathbf{R}\left(\sum_{i=1}^{m} \beta_i \eta_i + \sum_{j=1}^{p} \gamma_j \eta'_j\right) = \alpha\left(\sum_{i=1}^{m} \beta_i \mathbf{R}\eta_i + \underbrace{\sum_{j=1}^{p} \gamma_j \mathbf{R}\eta'_j}_{\text{Kernel term goes to zero}}\right) = \alpha \sum_{i=1}^{m} \beta_i \lambda_i \eta_i$$

i^{th} eigenvalue

Searching the Maximal Eigendirection

- $\hat{W}$ represents an unstable equilibrium
- Dominant direction of movement is the one corresponding to the largest eigenvalue, and these components must therefore grow in time
- Weight vector magnitude w grows indefinitely
- Direction approaches the eigenvector corresponding to the largest eigenvalue

$$\underbrace{E[\Delta W_k] = \alpha \sum_{i=1}^{m} \beta_i \lambda_i \eta_i}$$

Small perturbations cause weight changes to occur in directions away from that of $\hat{W}$ towards eigenvectors corresponding to non-zero eigenvalues

Oja's Rule

- Modification to the simple Hebbian weight change procedure

$$W_{k+1} = W_k + \alpha s_k (X_k - s_k W_k)$$
$$= W_k + \alpha s_k X'_k$$

Can be re-cast into a different form to clearly see the normalization

$$w_i^{k+1} = \frac{w_i^k + \alpha s_k x_i^k}{\sqrt{\sum_{j=1}^{n} \left(w_j^k + \alpha s_k x_j^k\right)^2}}$$

Re-compute the Average Weight Change

- Compute the expected weight change conditional on W_k

$$E[\Delta W_k] = E[s_k X_k - s_k^2 W_k] \\ = \mathbf{R} W_k - (W_k^T \mathbf{R} W_k) W_k$$

- Setting $E[W_k]$ to zero yields the equilibrium weight vector $\hat{W}$

$$E[\Delta W_k] = 0 = \mathbf{R}\hat{W} - (\hat{W}^T \mathbf{R} \hat{W})\hat{W}$$

$$\mathbf{R}\hat{W} = (\hat{W}^T \mathbf{R} \hat{W})\hat{W} = \lambda \hat{W}$$

- Define $\lambda = \hat{W}^T \mathbf{R} \hat{W} = \hat{W}^T \lambda \hat{W} = \lambda \hat{W}^T \hat{W} = \lambda \|\hat{W}\|^2$

- Shows that $\|\hat{W}\|^2 = 1$ → Self-normalizing!

Maximal Eigendirection is the only stable direction...

- Conducting a small neighbourhood analysis as before:

$$W = \eta_i + \epsilon$$

- Then the average weight change is:

$$E[\Delta W] = \mathbf{R}\epsilon - 2\lambda_i(\epsilon^T \eta_i)\eta_i - \lambda_i \epsilon + \mathcal{O}(\epsilon)$$

- Compute the component of the average weight change $E[\Delta W]$ along any *other* eigenvector, η_j for $j \neq i$

$$\eta_j^T E[\Delta W] = \underline{(\lambda_j - \lambda_i)}\eta_j^T \epsilon$$

clearly shows that the perturbation component along η_j must grow if $\lambda_j > \lambda_i$

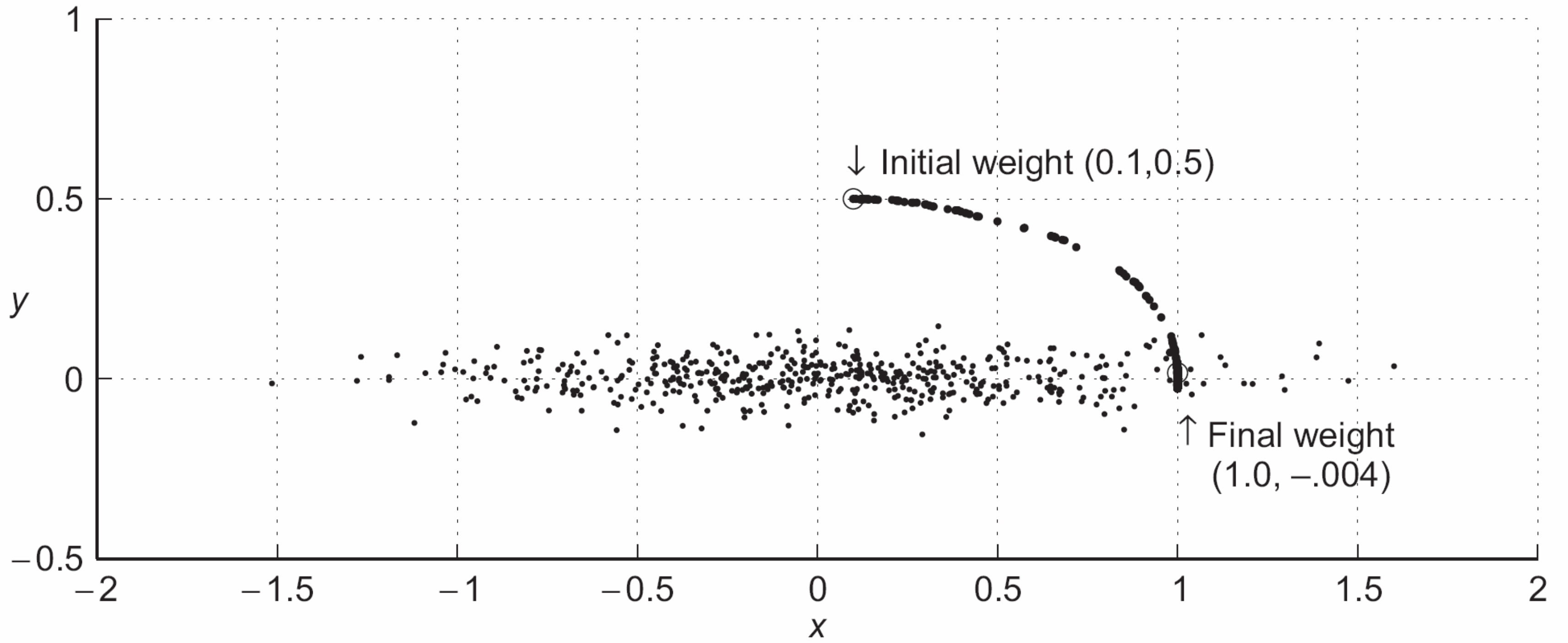
Operational Summary for Simulation of Oja's rule

Given	A set of feature vectors $\mathcal{X} = \{X_i\}$ drawn from a stationary stochastic distribution $p(X)$.
Initialize	↬ Weight vector $W \in \mathbb{R}^n$ of a linear neuron to some small random number (no bias required) ↬ Learning rate α to a small value (say $0.05 - 0.1$) ↬ Average weight change tolerance ϵ
Iterate	↻Repeat { ⇝ Pick an X_k with uniform probability over $\mathcal{X}$ ⇝ Compute the neuron signal $s_k = y_k = X_k^T W_k$ ⇝ Update weights: $W_{k+1} = W_k + \alpha s_k (X_k - s W_k)$ } until (the average weight change $E[\Delta W] < \epsilon$)

MATLAB Code for Oja's Rule

```
x=0.5*randn(500,1);    % Generate X scatter
y=0.05*randn(500,1); % Generate Y scatter
input=[x';y'];             % Create input data
        matrix
theta = 0;
r=[cos(theta) -sin(theta)
   sin(theta) cos(theta)];
inputnew=r*input;

xshift=0;
yshift=0;

for i=1:500
   inputnew(1,i)=inputnew(1,i)+xshift;
   inputnew(2,i)=inputnew(2,i)+yshift;
End

figure
hold on
zoom on
plot(inputnew(1,:),inputnew(2,:),'.k'); % Plot
        scatter
axis equal
grid on

eta=0.15;                  % Initialize learning rate
w=[.1;.5];                 % and the weights


for epoch=1:15          % We'll do 15 epochs
   for i=1:500

     % Compute activation
      s = inputnew(:,i)' * w;

      % Update weights
      w = w + eta * s * (input(:,i) - s*w);

      % Plot weight point
      plot(w(1),w(2),'.','markersize',10); end
end
```

Simulation of Oja's Rule



Principal Components Analysis

- Eigenvectors of the correlation matrix of the input data stream characterize the properties of the data set
- Represent *principal component directions* (orthogonal directions) in the input space that account for the data's variance
- High dimensional applications:
  - possible to neglect information in certain less important directions
  - retaining the information along other more important ones
  - reconstruct the data points to well within an acceptable error tolerance.

Subspace Decomposition

- To reduce dimension
  - Analyze the correlation matrix **R** of the data stream to find its eigenvectors and eigenvalues
  - Project the data onto the eigendirections.
  - Discard n-m components corresponding to n-m smallest eigenvalues

Sanger's Rule

- m node linear neuron network that accepts n-dimensional inputs can extract the first m principal components

$$\Delta w_{ij} = \alpha s_j (x_i - \sum_{k=1}^{j} s_k w_{ik}) \qquad j = 1, \ldots, m$$

- Sanger's rule reduces to Oja's learning rule for a single neuron
- Searches the first (and maximal) eigenvector or first principal component of the input data stream
- Weight vectors of the m units converge to the first *m* eigenvectors that correspond to eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$

Generalized Learning Laws

- Generalized *forgetting laws* take the form:

$$\frac{dW}{dt} = \phi(s)X - \gamma(s)W$$

- Assume that the impinging input vector $X \in \Re^n$ is a stochastic variable with stationary stochastic properties; $W \in \Re^n$ is the neuronal weight vector, and $\varphi(\cdot)$ and $\gamma(\cdot)$ are possibly non-linear functions of the neuronal signal
- Assume X is independent of W

Questions to Address

- What kind of information does the weight vector asymptotically encode?
- How does this information depend on the generalized functions $\varphi(\cdot)$ and $\gamma(\cdot)$?

Two Laws to Analyze

- Adaptation Law 1
  - A simple passive decay of weights proportional to the signal, and a reinforcement proportional to the external input:

$$\dot{W} = -\alpha s W + \beta X$$

- Adaptation Law 2
  - The standard Hebbian form of adaptation with signal driven passive weight decay:

$$\dot{W} = -\alpha s W + \beta s X$$

Analysis of Adaptation Law 1

$$\dot{W} = -\alpha s W + \beta X$$

- Since X is stochastic (with stationary properties), we are interested in the averaged or expected trajectory of the weight vector W
- Taking the expectation of both sides:

$$E\left[\dot{W} \mid W\right] = E\left[-\alpha s W + \beta X \mid W\right]$$

An Intermediate Result

$$\frac{1}{2}\frac{d}{dt}(W^T W) = \frac{1}{2}\frac{d}{dt}(\|W\|^2) = \|W\|\frac{d\|W\|}{dt} = W^T \dot{W}$$

$$= W^T(-\alpha s W + \beta X)$$

$$= -\alpha s \|W\|^2 + \beta X^T W$$

$$= s(\beta - \alpha\|W\|^2)$$

Asymptotic Analysis

- Note that the mean $\bar{X}$ is a constant
- We are interested in the average angle between the weight vector and the mean:

$$E\left[\frac{d\cos\theta}{dt}\right] = E\left[\frac{d}{dt}\left(\frac{\bar{X}^T W}{\|\bar{X}\|\|W\|}\right)\right]$$

$$= E\left[\frac{\bar{X}^T \dot{W}}{\|\bar{X}\|\|W\|} - \frac{\bar{X}^T W}{\|\bar{X}\|\|W\|^2}\frac{d\|W\|}{dt}\right]$$

Asymptotic Analysis

$$E\left[\frac{d\cos\theta}{dt}\right] = E\left[\frac{\bar{X}^T(-\alpha s W + \beta X)}{\|\bar{X}\|\|W\|} - \frac{(\bar{X}^T W)(-\alpha s\|W\|^2 + \beta X^T W)}{\|\bar{X}\|\|W\|^3}\right]$$

$$= \frac{\beta\|\bar{X}\|}{\|W\|} - \frac{\beta(\bar{X}^T W)^2}{\|\bar{X}\|\|W\|^3}$$

$$= \frac{\beta}{\|\bar{X}\|\|W\|^3}\left(\|\bar{X}\|^2\|W\|^2 - (\bar{X}^T W)^2\right)$$

$$\geq 0$$

where in the end we have employed the Cauchy–Schwarz inequality. Since *d*cos*θ/dt* is non-negative, *θ* converges uniformly to zero, with *d*cos*θ/dt* = 0 iff $\bar{X}$ and W have the same direction. Therefore, for finite $\bar{X}$ and W, the weight vector direction converges asymptotically to the direction of $\bar{X}$.

Analysis of Adaptation Law 2

$$\begin{aligned}\dot{W} &= -\alpha s W + \beta s X \\ &= -\alpha s W + \beta X s \\ &= -\alpha X^T W W + \beta X X^T W\end{aligned}$$

- Taking the expectation of both sides conditional on W

$$E[\dot{W}] = -\alpha \bar{X}^T W W + \beta \mathbf{R} W$$

Fixed points of W

- To find the fixed points, set the expectation of the expected weight derivative to zero:

$$E[\dot{W}] = 0 = -\alpha \bar{X}^T \hat{W} \hat{W} + \beta \mathbf{R} \hat{W}$$

- From where

$$\mathbf{R}\hat{W} = \left(\frac{\alpha \bar{X}^T \hat{W}}{\beta} \right) \hat{W} = \lambda \hat{W}$$

- Clearly, eigenvectors of **R** are fixed point solutions of W

All Eigensolutions are not Stable

- The i^{th} solution is the eigenvector η_i of $\mathbf{R}$ with corresponding eigenvalue

$$\lambda_i = \frac{\alpha \bar{X}^T \eta_i}{\beta}$$

- Define θ_i as the angle between W and η_i, and analyze (as before) the average value of rate of change of $\cos \theta_i$, conditional on W

Asymptotic Analysis

$$E\left[\frac{d\cos\theta_i}{dt}\right] = E\left[\frac{d}{dt}\left(\frac{\eta_i^T W}{\|\eta_i\|\|W\|}\right)\right]$$

$$= E\left[\frac{\eta_i^T \dot{W}}{\|\eta_i\|\|W\|} - \frac{\eta_i^T W}{\|\eta_i\|\|W\|^2}\frac{d}{dt}\|W\|\right]$$

$$= E\left[\frac{\eta_i^T \dot{W}}{\|\eta_i\|\|W\|} - \frac{\eta_i^T W W^T \dot{W}}{\|\eta_i\|\|W\|^3}\right]$$

$$= \frac{\eta_i^T(-\alpha \bar{X}^T W W + \beta \mathbf{R} W)}{\|\eta_i\|\|W\|} - \frac{\eta_i^T W W^T(-\alpha \bar{X}^T W W + \beta \mathbf{R} W)}{\|\eta_i\|\|W\|^3}$$

Contd.

Asymptotic Analysis

$$E\left[\frac{d\cos\theta_i}{dt}\right] = \frac{\beta\eta_i^T \mathbf{R} W}{\|\eta_i\|\|W\|} - \frac{\beta\eta_i^T W(W^T \mathbf{R} W)}{\|\eta_i\|\|W\|^3}$$

$$= \frac{\beta\lambda_i\eta_i^T W}{\|\eta_i\|\|W\|} - \frac{\beta\eta_i^T W(W^T \mathbf{R} W)}{\|\eta_i\|\|W\|^3}$$

$$= \frac{\beta\eta_i^T W}{\|\eta_i\|\|W\|}\left(\lambda_i - \frac{W^T \mathbf{R} W}{\|W\|^2}\right)$$

$$= \beta\cos\theta_i\left(\lambda_i - \frac{W^T \mathbf{R} W}{\|W\|^2}\right)$$

Asymptotic Analysis

- It follows from the Rayleigh quotient that the parenthetic term is guaranteed to be positive only for $\lambda_i = \lambda_{max}$, which means that for the eigenvector η_{max} the angle θ_{max} between W and η_{max} monotonically tends to zero as learning proceeds

$$E\left[\frac{d\cos\theta_i}{dt}\right]$$

$$= \beta\cos\theta_i\left(\lambda_i - \frac{W^T\mathbf{R}W}{\|W\|^2}\right)$$

First Limit Theorem

- Let $\alpha > 0$, and $s = X^T W$. Let $\gamma(s)$ be an arbitrary scalar function of s such that $E[\gamma(s)]$ exists. Let $X(t) \in \Re^n$ be a stochastic vector with stationary stochastic properties, $\bar{X}$ being the mean of $X(t)$ and $X(t)$ being independent of W
- If equations of the form

$$\dot{W} = E[\alpha X - \gamma(s) W]$$

have non-zero bounded asymptotic solutions, then these solutions must have the same direction as that of $\bar{X}$

Second Limit Theorem

- Let α, s and $\gamma(s)$ be the same as in Limit Theorem 1. Let $\mathbf{R} = E[XX^T]$ be the correlation matrix of X. If equations of the form :

$$\dot{W} = E[\alpha s X - \gamma(s) W]$$

have non-zero bounded asymptotic solutions, then these solutions must have the same direction as η_{max} where η_{max}, is the maximal eigenvector of $\mathbf{R}$ with eigenvalue λ_{max}, provided $\eta^T_{max} W(0) = 0$

Competitive Neural Networks

- Competitive networks
  - cluster
  - encode
  - classify

  data by identifying
  - vectors which logically belong to the same category
  - vectors that share similar properties
- Competitive learning algorithms use competition between lateral neurons in a layer (via lateral interconnections) to provide selectivity (or localization) of the learning process

Types of Competition

- *Hard competition*
  - exactly one neuron—the one with the largest activation in the layer—is declared the winner
  - ART 1 F_2 layer
- *Soft competition*
  - competition suppresses the activities of all neurons except those that might lie in a neighbourhood of the true winner
  - Mexican Hat Nets

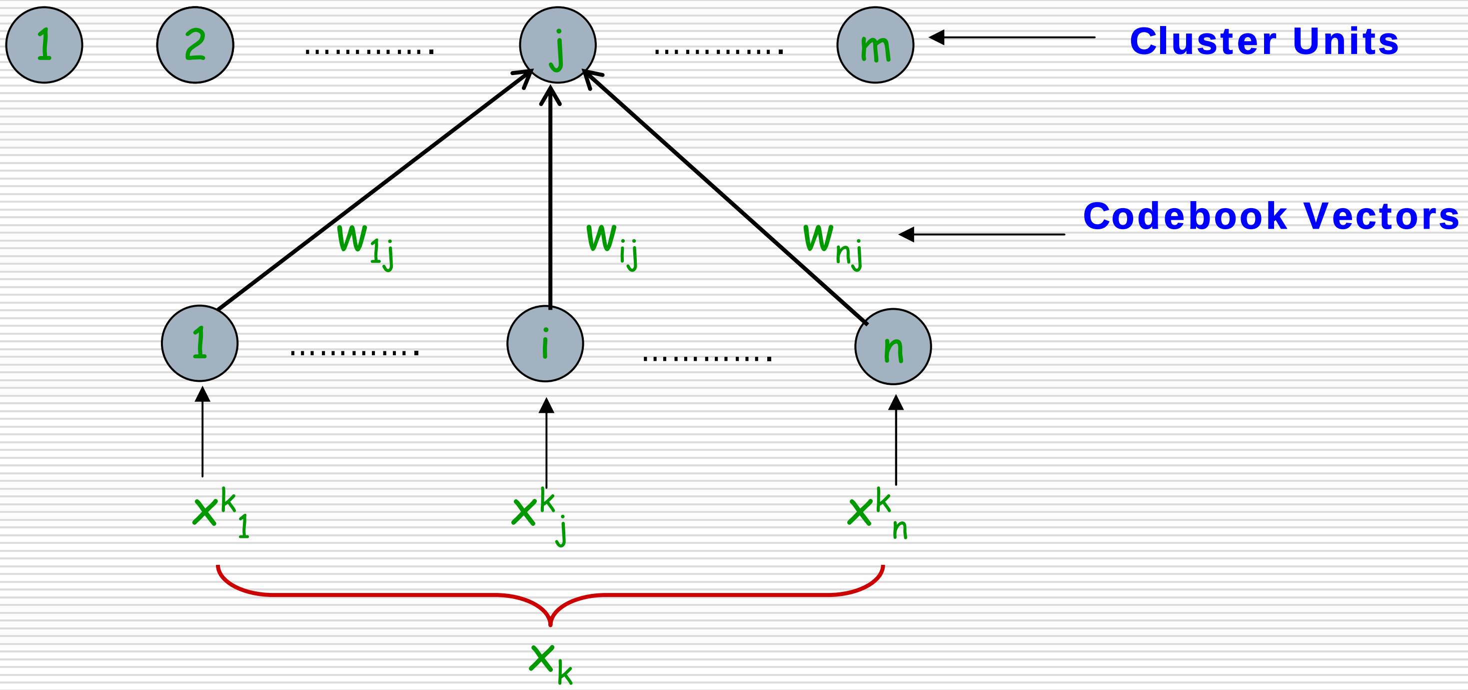
Competitive Learning is Localized

- CL algorithms employ *localized learning*
  - update weights of only the active neuron(s)
- CL algorithms identify *codebook vectors* that represent invariant features of a cluster or class

Vector Quantization

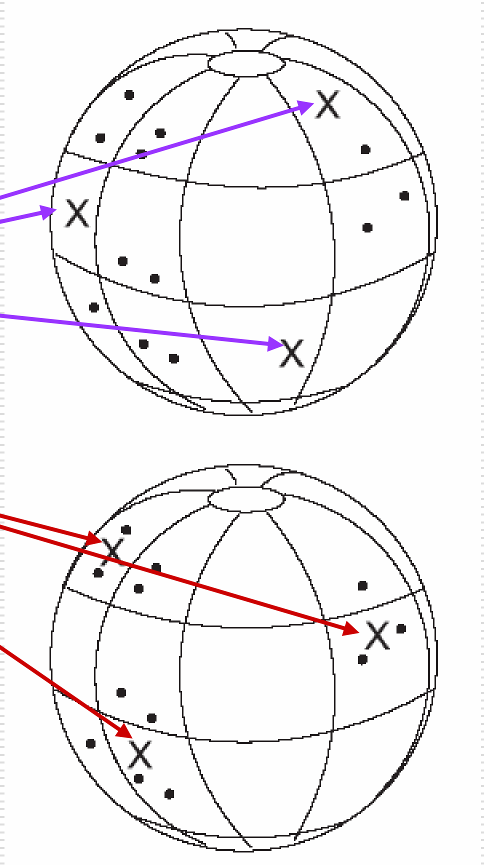
- If many patterns X_k cause cluster neuron j to fire with maximum activation a codebook vector $W_j = (w_{1j}, \ldots, w_{nj})^T$ behaves like a quantizing vector
- Quantizing vector : representative of all members of the cluster or class
- This process of representation is called vector quantization
- Principal Applications
  - signal compression
  - function approximation
  - image processing

Competitive Learning Network



Example of CL

- Three clusters of vectors (denoted by solid dots) distributed on the unit sphere
- Initially randomized codebook vectors (crosses) move under influence of a competitive learning rule to approximate the centroids of the clusters
- Competitive learning schemes use codebook vectors to approximate centroids of data clusters



Principle of Competitive Learning

- Given a sequence of stochastic vectors $X_k \in \Re^n$ drawn from a possibly unknown distribution, each pattern X_k is compared with a set of initially randomized weight vectors $W_j \in \Re^n$ and the vector W_J which best matches X_k is to be updated to match X_k more closely

Inner Product vs Euclidean Distance Based Competition

- Inner Product

$$y_J = \max_j \{ X_k^T W_j \}$$

- Euclidean Distance Based Competition

$$\| X_k - W_J \| = \min_j \{ \| X_k - W_j \| \}$$

Two sides of the same coin!

- Assume: weight vector equinorm property $\|W_1\| = \|W_2\| = \ldots = \|W_m\|$

$$\|X_k - W_J\|^2 = \min_j \{\|X_k - W_j\|^2\}$$

$$\Rightarrow \quad (X_k - W_J)^T (X_k - W_J) = \min_j \{(X_k - W_j)^T (X_k - W_j)\}$$

$$\Rightarrow \quad \|X_k\|^2 - 2X_k^T W_J + \|W_J\|^2 = \min_j \{(\|X_k\|^2 - 2X_k^T W_j + \|W_j\|^2)\}$$

$$\Rightarrow \quad -2X_k^T W_J + \|W_J\|^2 = \min_j \{(-2X_k^T W_j + \|W_j\|^2)\}$$

$$-2X_k^T W_J = \min_j \{-2X_k^T W_j\}$$

$$X_k^T W_J = \max_j \{X_k^T W_j\}$$

Generalized CL Law

- For an n - neuron competitive network

$$w_{ij}^{k+1} = \begin{cases} w_{ij}^k + \eta(x_i^k - w_{ij}^k) & i = 1, \ldots, n \quad j = J \\ w_{ij}^k & i = 1, \ldots, n \quad j \neq J \end{cases}$$

$$J = \arg\max_j \{y_j^k\}$$

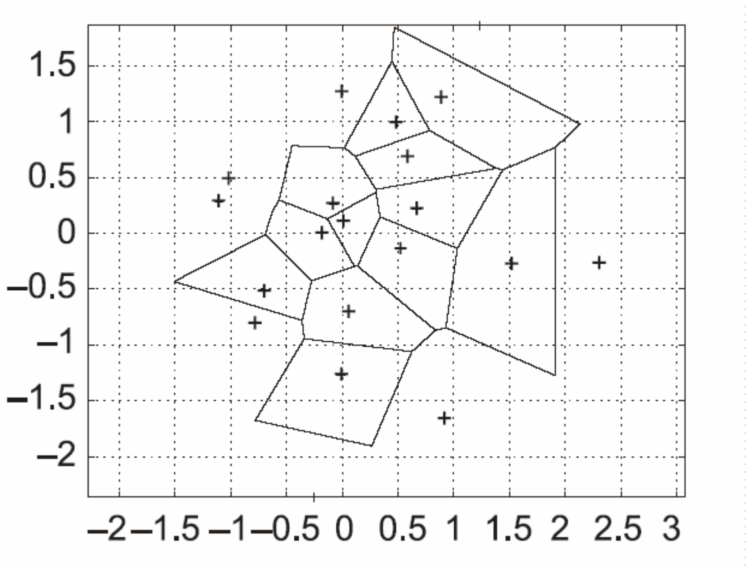
Vector Quantization Revisited

- An important application of competitive learning
- Originally developed for information compression applications
- Routinely employed to store and transmit speech or vision data.
- VQ places codebook vectors W_i into the signal space in a way that minimizes the expected quantization error

$$E = \int \|X - W_J\|^2 p(X) dX$$

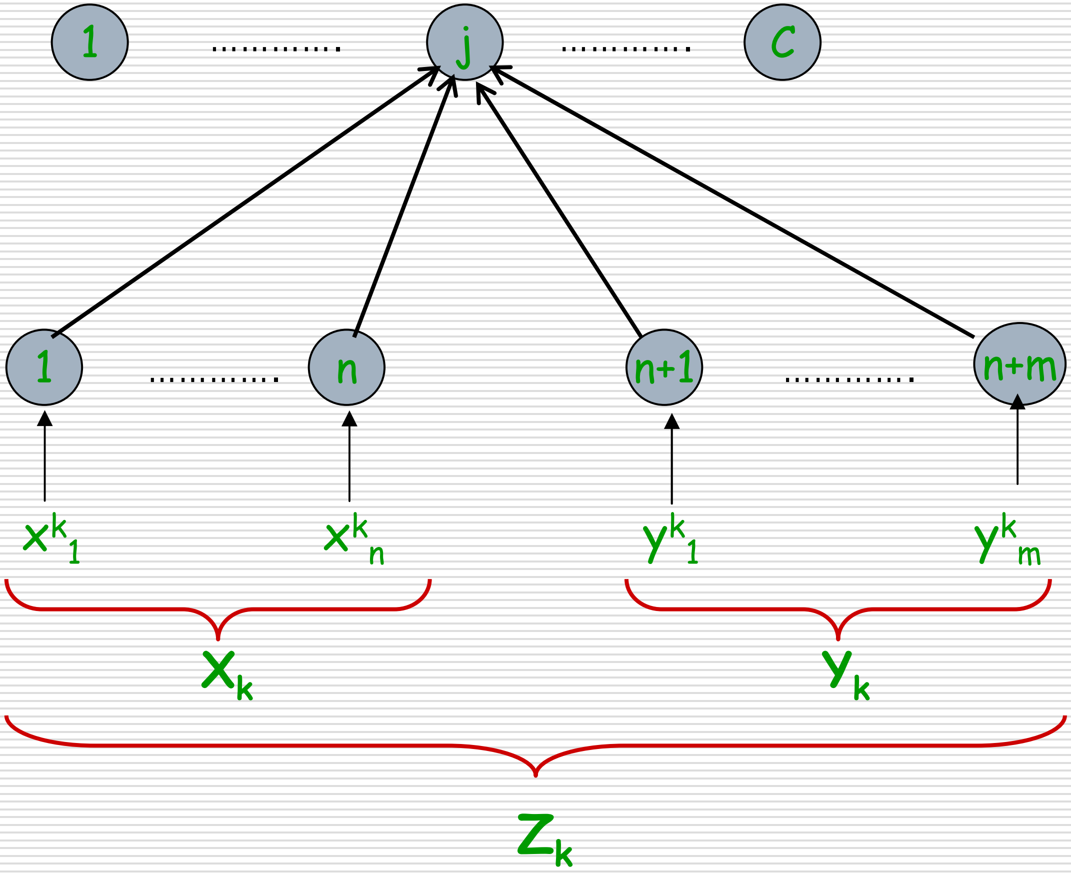
Example: Voronoi Tesselation

- Depict classification regions that are formed using the 1-nearest neighbour classification rule
- Voronoi bin specified by a codebook vector W_J is simply the set of points in R^n whose nearest neighbour of all W_j is W_J a Euclidean distance measure



20 randomly generated Gaussian distributed points using the MATLAB **voronoi** command

Unsupervised Vector Quantization



Unsupervised VQ

- Compares the current random sample vector $Z_k = (X_k \mid Y_k)$ with the C quantizing weight vectors $W_j\ (k)$ (weight vector W_j at time instant k)
- Neuron J wins based on a standard Euclidean distance competition

$$\| W_{J(k)} - Z_k \| = \min_j \| W_{j(k)} - Z_k \|$$

Unsupervised VQ Learning

- Neuron J learns the input pattern in accordance with standard competitive learning in vector form:

$$W_{J(k+1)} = W_{J(k)} + \eta_k (Z_k - W_{J(k)})$$

- Learning coefficient η_k should decrease gradually towards zero
- Example: $\eta_k = \eta_0[1 - k/2Q]$ for an initial learning rate η_0 and Q training samples
- Makes η decrease linearly from η_0 to zero over $2Q$ iterations

Scaling the Data Components

- Scale data samples $\{Z_k\}$ such that all features have equal weight in the distance measure
- Ensures that no one variable dominates the choice of the winner
- Embedded within the distance computation:

$$\left(W_{J(k)} - Z_k\right)^T \Omega \left(W_{J(k)} - Z_k\right) = \min_j \left\{ \left(W_{j(k)} - Z_k\right)^T \Omega \left(W_{j(k)} - Z_k\right) \right\}$$

$$\Omega = \begin{bmatrix} \omega_1^2 & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \omega_{m+n}^2 \end{bmatrix}$$

Operational Summary of AVQ

Given	Concatenated input data, $\{Z_k\}_{k=1}^{Q}$, $Z_k \in \mathbb{R}^{n+m}$, preprocessed for normalization and scaling.
Initialize	$\looparrowright$ Number of clusters C to be generated. $\looparrowright$ Quantization vectors W_{j0} of all C clusters to random samples of the input data. $\looparrowright$ Learning rate schedule: $\eta_k = 0.1\left(1-\left(\frac{k}{2Q}\right)\right)$, $\eta_0 = 0.1$ $\looparrowright$ Maximum number of iterations MAXITER = Q or $2Q$ $\looparrowright$ Iteration index $k = 0$.

Operational Summary of AVQ

Iterate ⟳Repeat

{

$\rightsquigarrow$ Pick a data sample Z_k from a random sequence

$\rightsquigarrow$ Find the winning neuron index J:

$\|W_{J(k)} - Z_k\| = \min_j\{\|W_{j(k)} - Z_k\|\}$

$\rightsquigarrow$ Update the winning neuron synapses:

$w_{iJ}^{k+1} = w_{iJ}^{k} + \eta_k(z_i^k - w_{iJ}^{k}) \quad i = 1, \ldots, n+m$

$\rightsquigarrow$ Update learning rate:

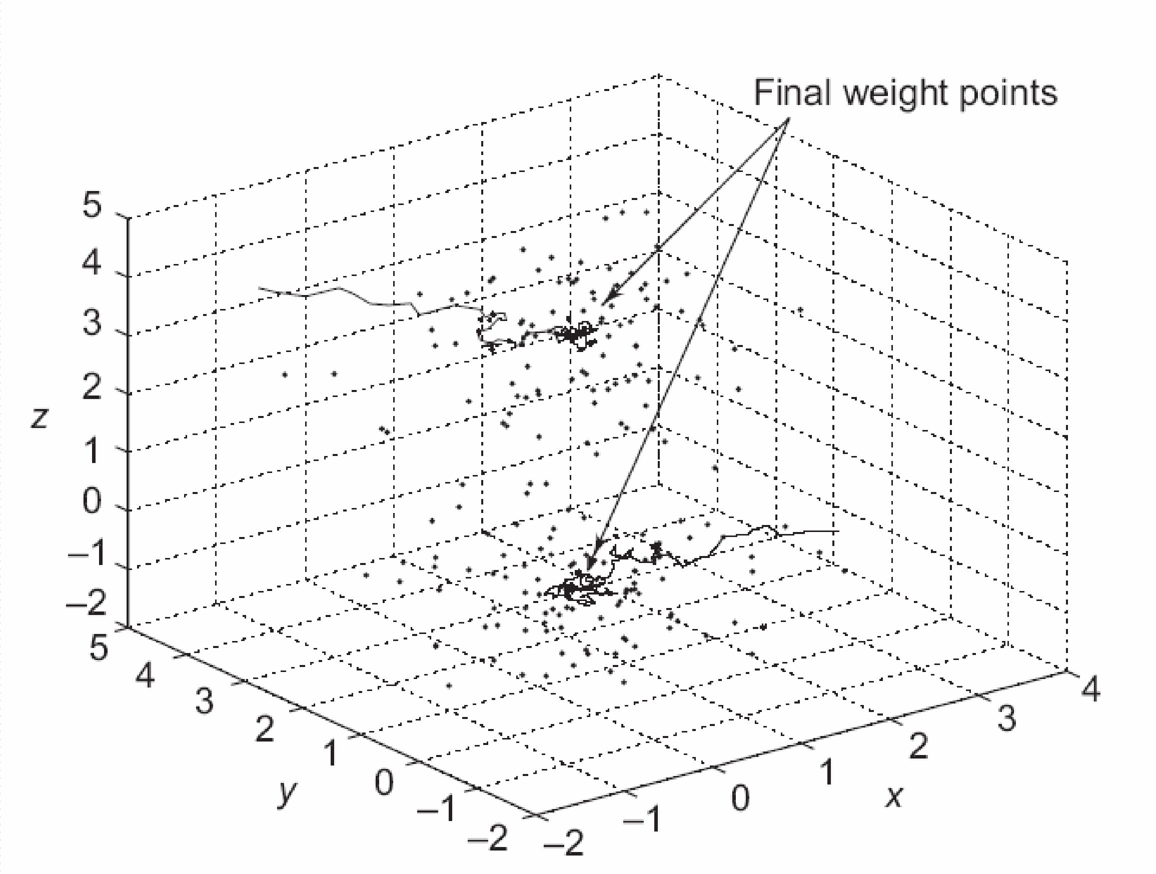
$\eta_k = 0.1\left(1 - \left(\frac{k}{2Q}\right)\right).$

} while (k < MAXITER)

MATLAB Simulation Example on AVQ

- Cluster a three dimensional data set comprising 200 data points using adaptive vector quantization
- Data points generated to be randomly and normally distributed: 100 data points each about centers with coordinates (0,0,0) and (1,2,3), with a standard deviation 0.8
- Cluster field assumed to comprise two neurons with instar weights initialized to (3,0,0) and (–2,3,5) respectively

MATLAB Simulation Example on AVQ



MATLAB Code for AVQ

```
% Program for AVQ Clustering: m clusters in n
        dimensions

m = 2; % Generate two clusters
fid=fopen('./avqtest.dat','r');% Open data file
pat = fscanf(fid,'%f %f',[3 inf]);
fclose(fid);

% dimension n, and number of data Q
[n,Q]=size(pat);

% Initial weight matrix w
w = [3 -2; 0 3; 0 5];

figure % plot the clusters
plot3(pat(1,:), pat(2,:), pat(3,:), 'b.');
grid on;
hold on;
axis([-2 4 -2 5 -2 5]);
```

```
for i=1:Q % for each data point
  % reset the minimum distance index
  minindex = -1;
  % set the mindist variable to a large number
  mindist = 1000;
    for j=1:m % check distance to each codebook
      dist = 0;
      for k=1:n
        dist = dist + (pat(k,i)-w(k,j))^2;
      end
      dist = sqrt(dist);
      if dist < mindist
        minindex = j;
        mindist = dist;
      end
    end
    eta = 0.1*(1-(i/(2*Q))); % Update learning rate
    for k=1:n % update the winning weight vector
    w(k,minindex) = w(k,minindex) + eta*(pat(k,i) –

      w(k,minindex));
end
```

Supervised Vector Quantization

- Suggested by Kohonen
- Uses a supervised version of vector quantization
  - *Learning vector quantization* (LVQ1)
- Data classes defined in advance and each data sample is labelled with its class

$$w_{iJ}^{k+1} = \begin{cases} w_{iJ}^{k} + \eta_k \left(x_i^k - w_{iJ}^{k}\right) & \text{if} \quad X_k \in \mathcal{C}_J \\ w_{iJ}^{k} - \eta_k \left(x_i^k - w_{iJ}^{k}\right) & \text{if} \quad X_k \notin \mathcal{C}_J \end{cases}$$

$$w_{ij}^{k+1} = w_{ij}^{k} \qquad \text{for} \quad j \neq J$$

Practical Aspects of LVQ1

- $0 < \eta_k < 1$ decreases monotonically with successive iterations
- Recommended that η_k be kept small: 0.1
- Vectors in a limited training set may be applied cyclically to the system as η_k is made to decrease linearly to zero
- Use an equal number of codebook vectors per class
  - Leads to an optimal approximation of the class borders
- Initialization of codebook vectors may be done to actual samples of each class
- Define the number of iterations in advance:
  - Anything from 50 to 200 times the number of codebook vectors selected for representation

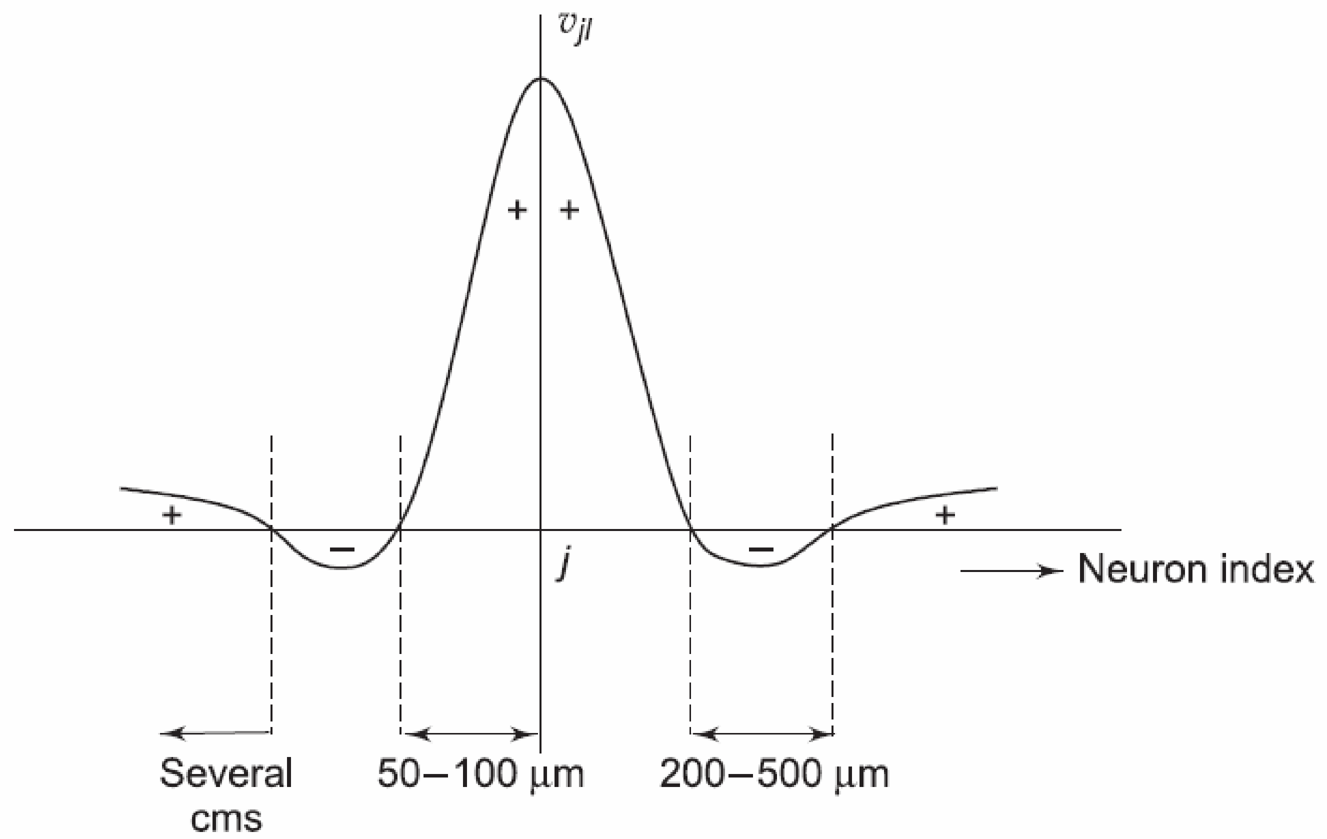
Operational Summary of LVQ1

Given	Input stream of labelled vectors $\{X_k\}_{k=1}^{Q}$ $X_k \in \mathbb{R}^n$ that belong to one of C classes $\{\mathcal{C}_j\}_{j=1}^{C}$
Initialize	↬ Number of classes C to be generated. ↬ Quantization vectors W_{j0} of all C classes to random samples of the input data. ↬ Learning rate schedule: $\eta_k = 0.1\big(1 - (\frac{k}{2Q})\big)$, $\eta_0 = 0.1$ ↬ Maximum number of iterations MAXITER = Q or $2Q$ ↬ Iteration index $k = 0$.
Iterate	↻Repeat { ⇝ Pick a data sample X_k from the data stream ⇝ Find the winning neuron index J: $\\|W_{J(k)} - X_k\\| = \min_j\{\\|W_{j(k)} - X_k\\|\}$ ⇝ Update only the winning neuron synapses: $w_{iJ}^{k+1} = w_{iJ}^{k} + \eta_k(x_i^k - w_{iJ}^{k})$ if $X_k \in \mathcal{C}_J$ $w_{iJ}^{k+1} = w_{iJ}^{k} - \eta_k(x_i^k - w_{iJ}^{k})$ if $X_k \notin \mathcal{C}_J$ ⇝ Update learning rate $\eta_k = 0.1\big(1 - (\frac{k}{2Q})\big)$. } while ($k <$ MAXITER)

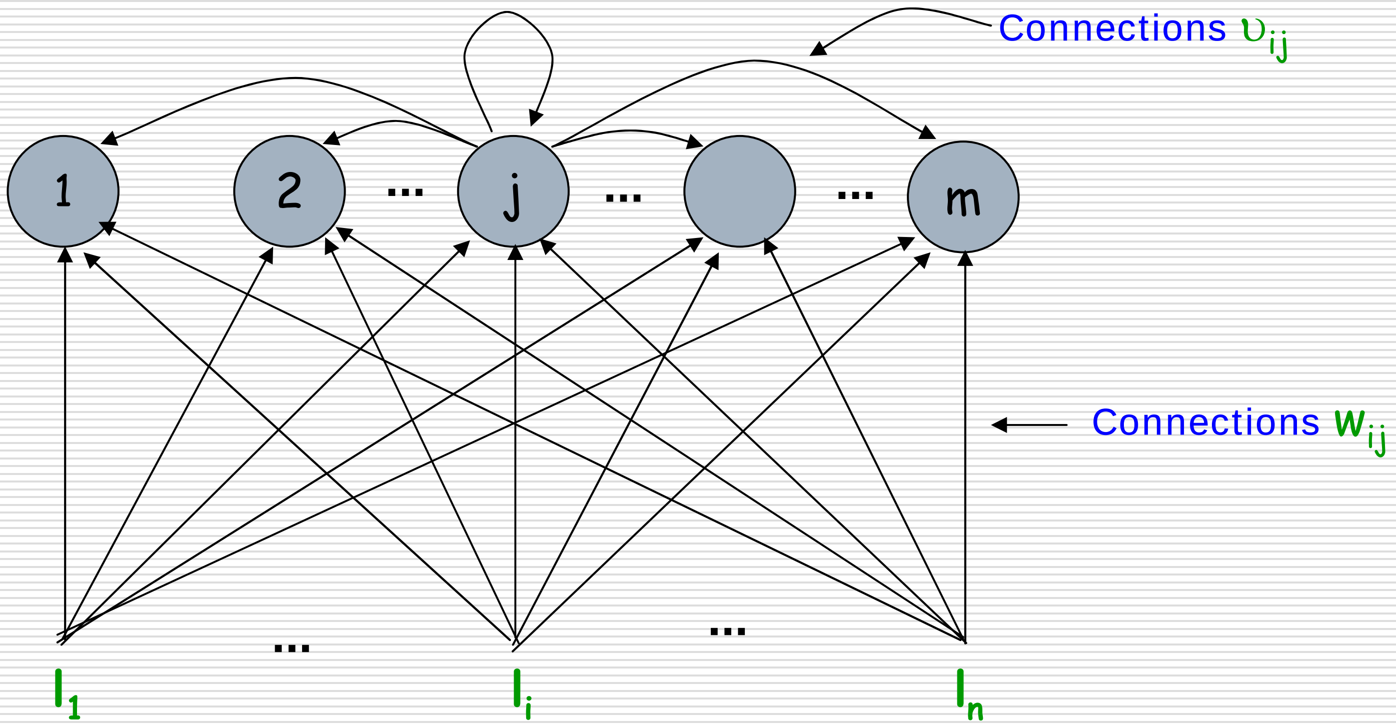
Mexican Hat Networks

- Closely follow biological structure
- Evidence that certain two-dimensional structures of visual cortex neurons have lateral interactions with a connectivity pattern that exhibits:
  - Short range lateral excitation within a radius of 50–100 μm
  - Region of inhibitory interactions outside the area of short range
  - Excitation which extends to a distance of about 200–500 μm

Mexican Hat Connectivity Pattern



Mexican Hat Neural Network



Mexican Hat Neural Network

- Every neuron in the network follows has Mexican Hat lateral connectivity
- Two distinguishing behavioural properties:
  - Spatial activity across the network clusters locally about winning neurons
  - Local cluster positions are decided by the nature of the input pattern

Mexican Hat Neural Network

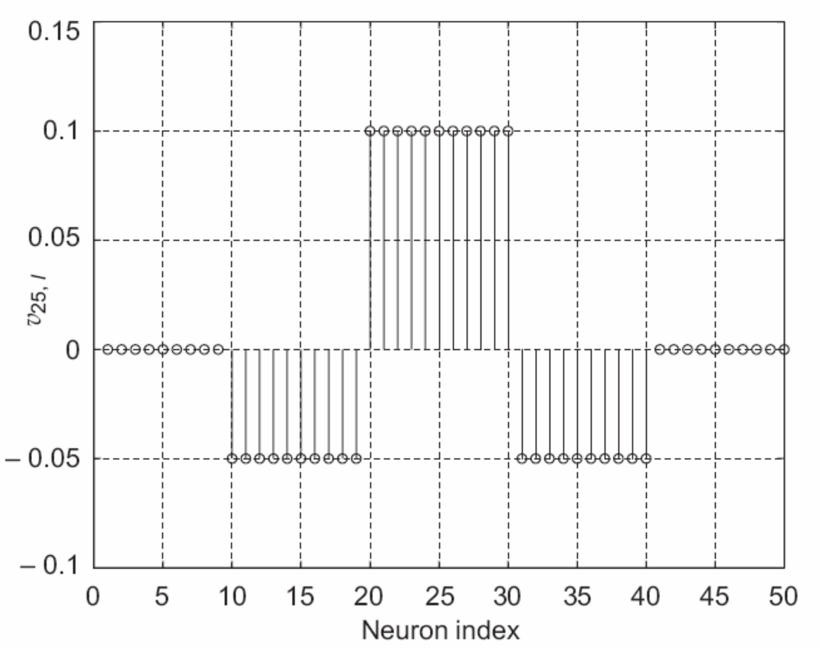
- Quantify the total neuronal activity for the j th neuron as a sum of two components:

$$x_j = \underbrace{\sum_{i=1}^{n} w_{ij} I_i}_{\text{External input}} + \underbrace{\sum_{l=1}^{m} v_{lj} s_l}_{\text{Lateral feedback}} \qquad j = 1, \ldots, m$$

Possibly non-linear signal function usually the piecewise linear threshold function

Discrete Approximation to Mexican Hat Connectivity

- Required for simulation
- A neuron receives
  - constant lateral excitation from 2L neighbours
  - constant lateral inhibition from 2M neighbours



One Dimensional Mexican Hat Network Simulation

- Assume that index i runs over values assuming neuron j to be centered at position 0
- Signals that correspond to index values that are out of range are simply to be disregarded (assumed zero)
- $I_j = \varphi(j)$ is a smooth function of the array index j

$$x_j = a \sum_{i=-L}^{L} s_i - b\left(\sum_{i=-L-M}^{-L-1} s_i + \sum_{i=L+1}^{L+M} s_i \right) + I_j \qquad j = 1, \dots, m$$

Generalized Difference Form

Note the introduction of time index **k**

$$x_j^{k+1} = \gamma\left\{a\sum_{i=-L}^{L}\mathcal{S}(x_i^k) - b\left(\sum_{i=-L-M}^{-L-1}\mathcal{S}(x_i^k) + \sum_{i=L+1}^{L+M}\mathcal{S}(x_i^k)\right)\right\} + I_j^k$$

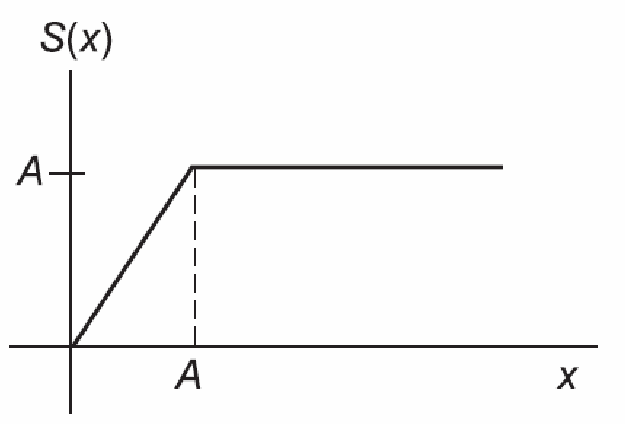
a, b control the extent of excitation and inhibition that a neuron receives

feedback factor **γ** determines the proportion of feedback that contributes to the new activation

Neuron Signal Function

- Uniformly assumed piecewise linear

$$\mathcal{S}(x) = \begin{cases} A & x \geqslant A \\ x & 0 \leq x < A \\ 0 & x < 0 \end{cases}$$

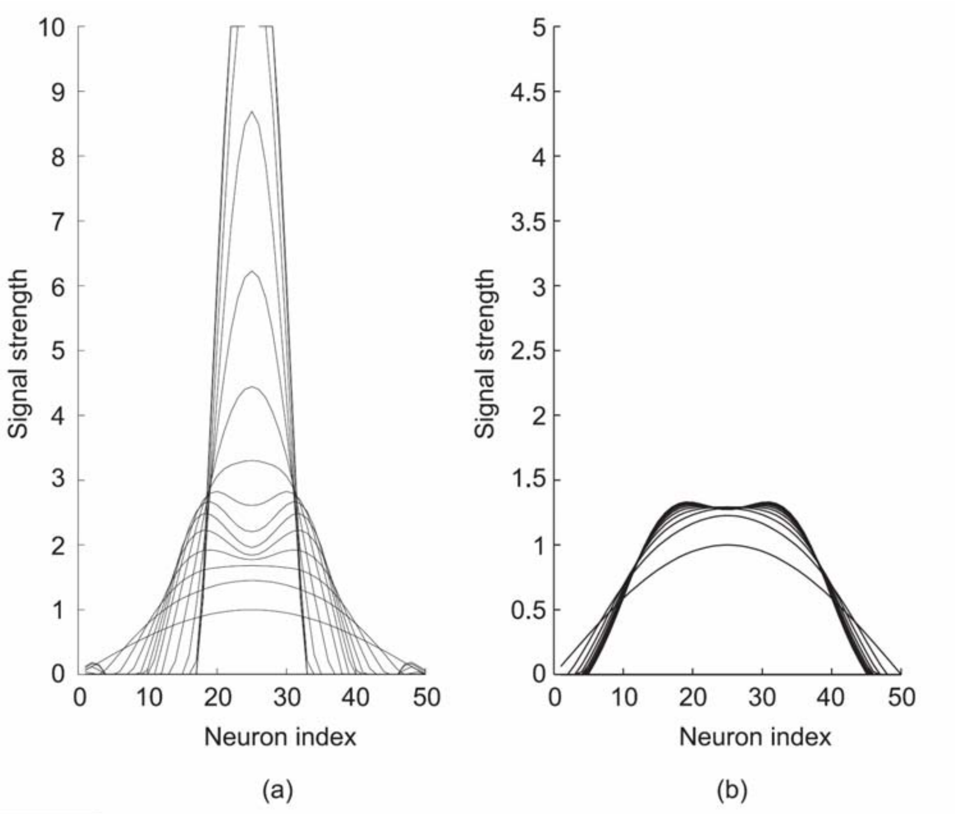


One Dimensional Simulation

- Assume a field of 50 linear threshold neurons
- Each has a discrete Mexican Hat connectivity pattern
- Simulate the system assuming a smooth sinusoidal input to the network:

$$I_i = \sin\left(\frac{\pi i}{50}\right) \qquad i = 1, \ldots, 50$$

One Dimensional Simulation



(a) 15 snapshots of neuron field updates with $\gamma = 1.5$.

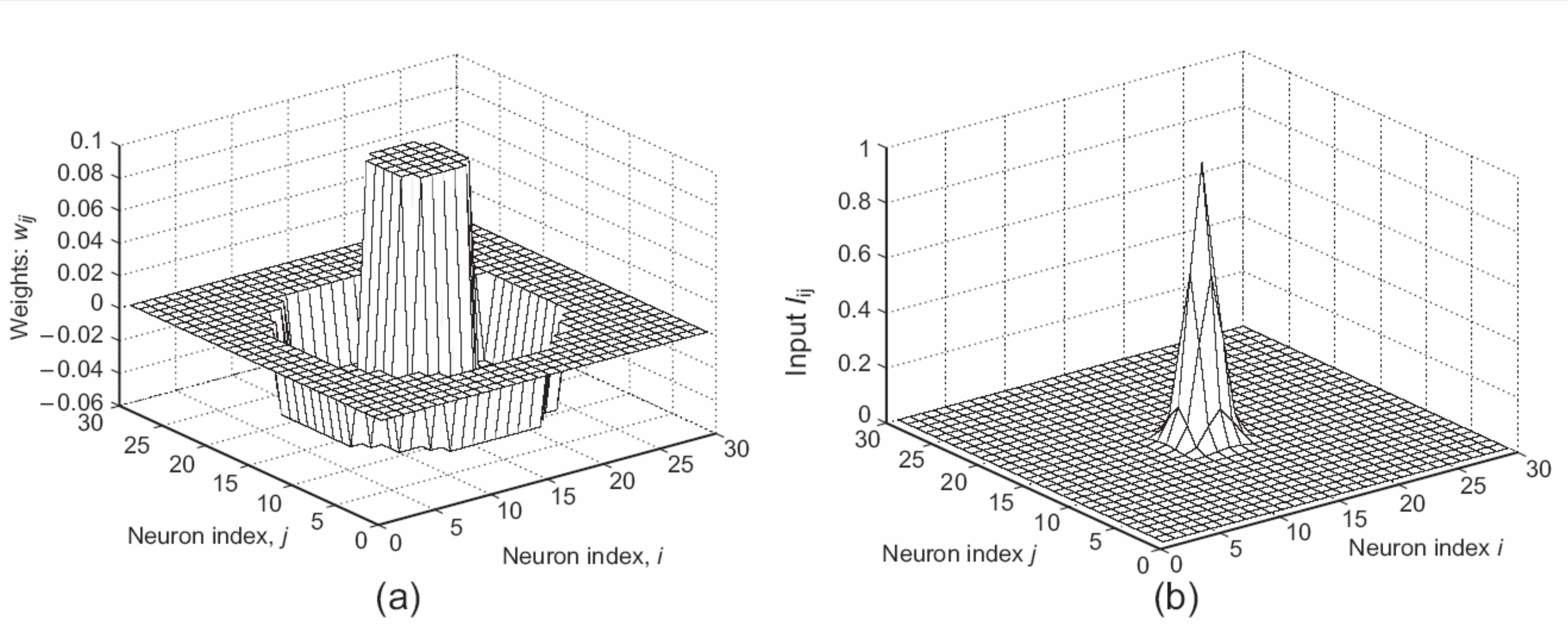
(b) 15 snapshots of neuron field updates with $\gamma = 0.75$

MATLAB Code for Mexican Hat Network

```
% Mexican Hat Network Simulation

leradius = 5; % excitation radius
liwidth = 10; % inhibition radius
interactlen = leradius + liwidth +1;
max = 10; % maximum signal value
excit = 0.1; % a
inhibit = -0.05;% b
feedback = 1.5; % gamma
for j=1:50 % Generate the Mex hat connectivity
  for i=1:50 % 50 x 50 weights
   indexdif = abs(j-i);
   if (indexdif < interactlen)
      if (indexdif < leradius+1)
        w(j,i) = excit;
      else
        w(j,i) = inhibit;
      end
    else
     w(j,i) = 0;
    end
  end
end
```

```
index=1:1:50;
input = sin(pi.*index/50);% set up input vector
s=zeros(50); % initialize signals
figure;
hold on

for t=1:15
  for i =1:50 % compute activations
    activation(i) = input(i);
    for j=1:50
      activation(i) = activation(i) +
                                              feedback*w(j,i)*s(j);
     end
  end
  for i=1:50 % compute signals
  if (activation(i) > max) s(i) = max;
  elseif (activation(i) < 0) s(i) = 0;
  else s(i) = activation(i);
  end
 end
 plot(index, s);
end
xlabel('Neuron index'); ylabel('Signal strength');
```

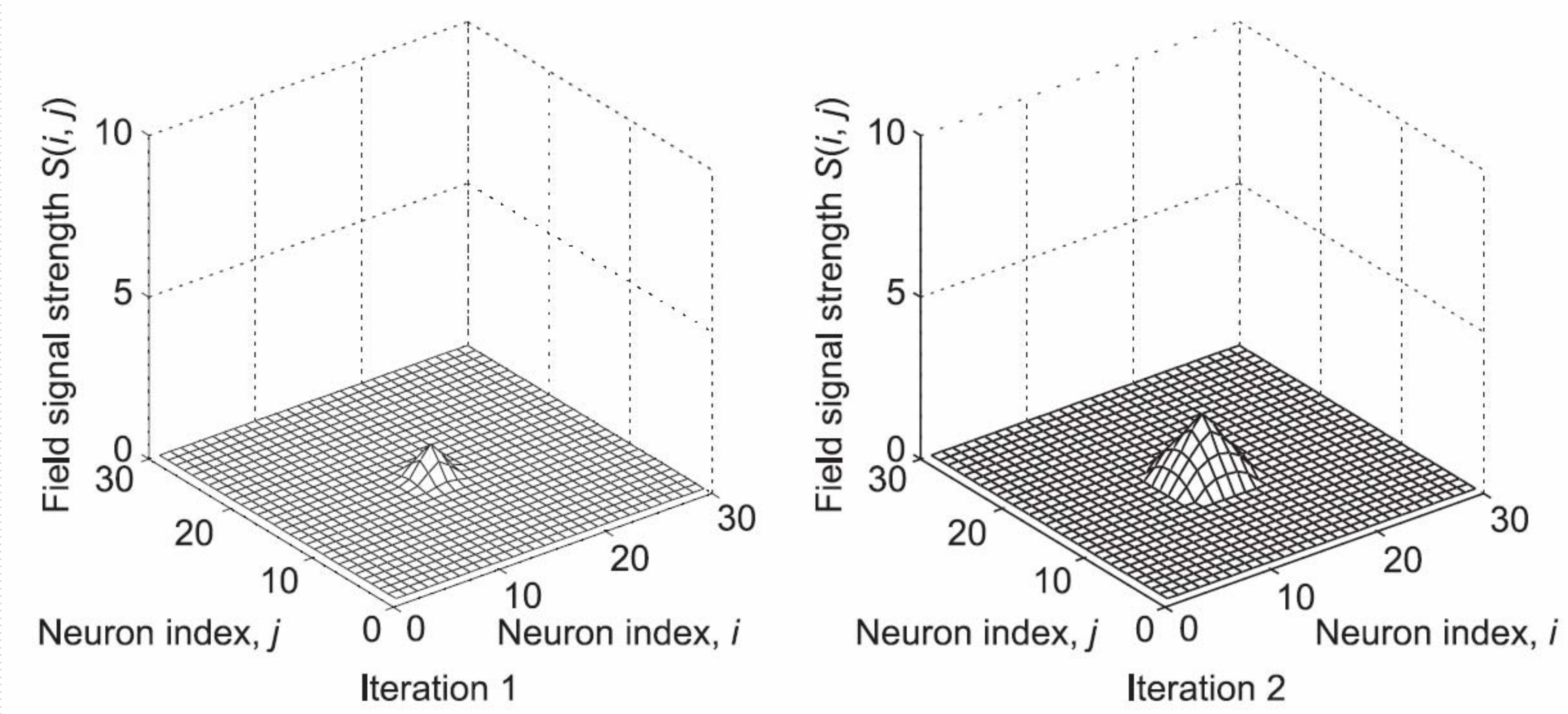
Two Dimensional Mexican Hat Network Simulation



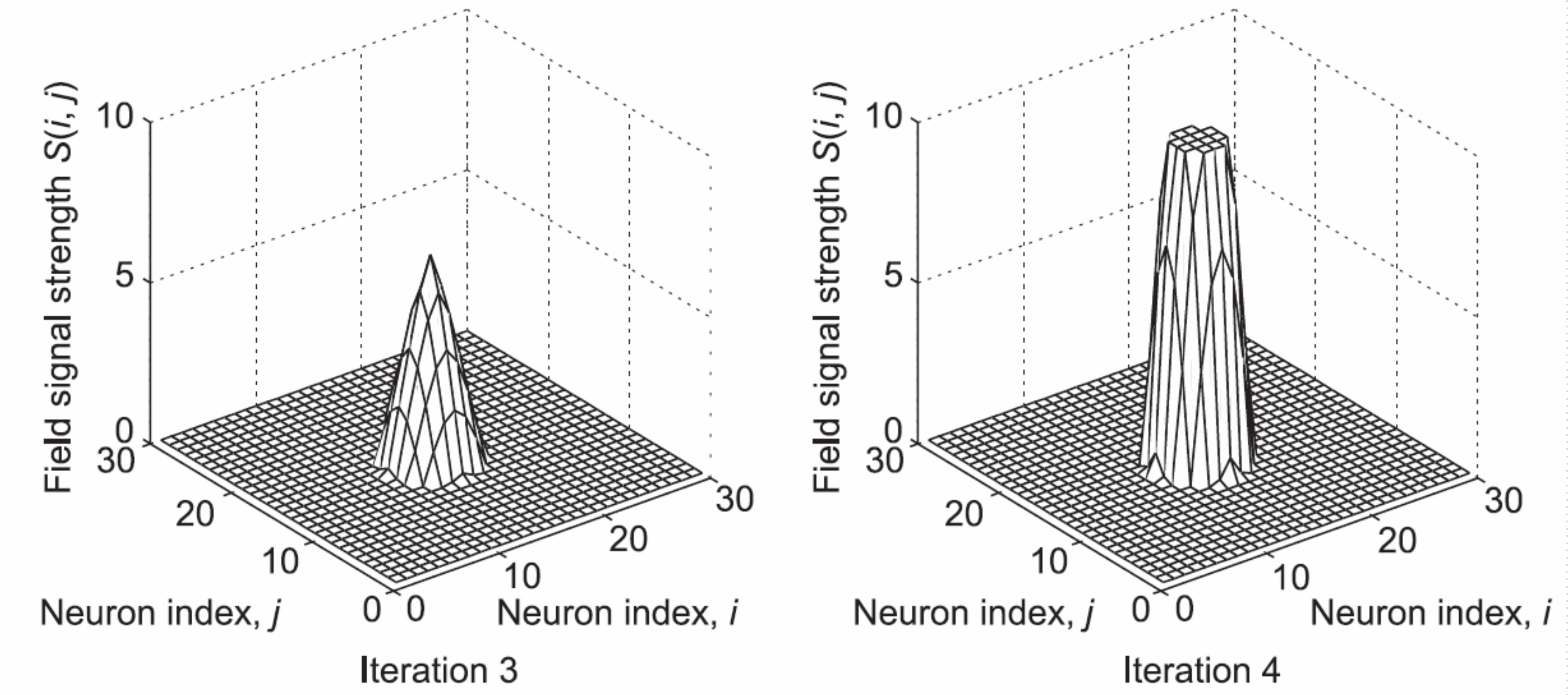
(a) Mexican hat connectivity portrayed for the central neuron in a 30 × 30 planar neuron field

(b) Two dimensional Gaussian input assumed for the simulation of the planar Mexican hat network

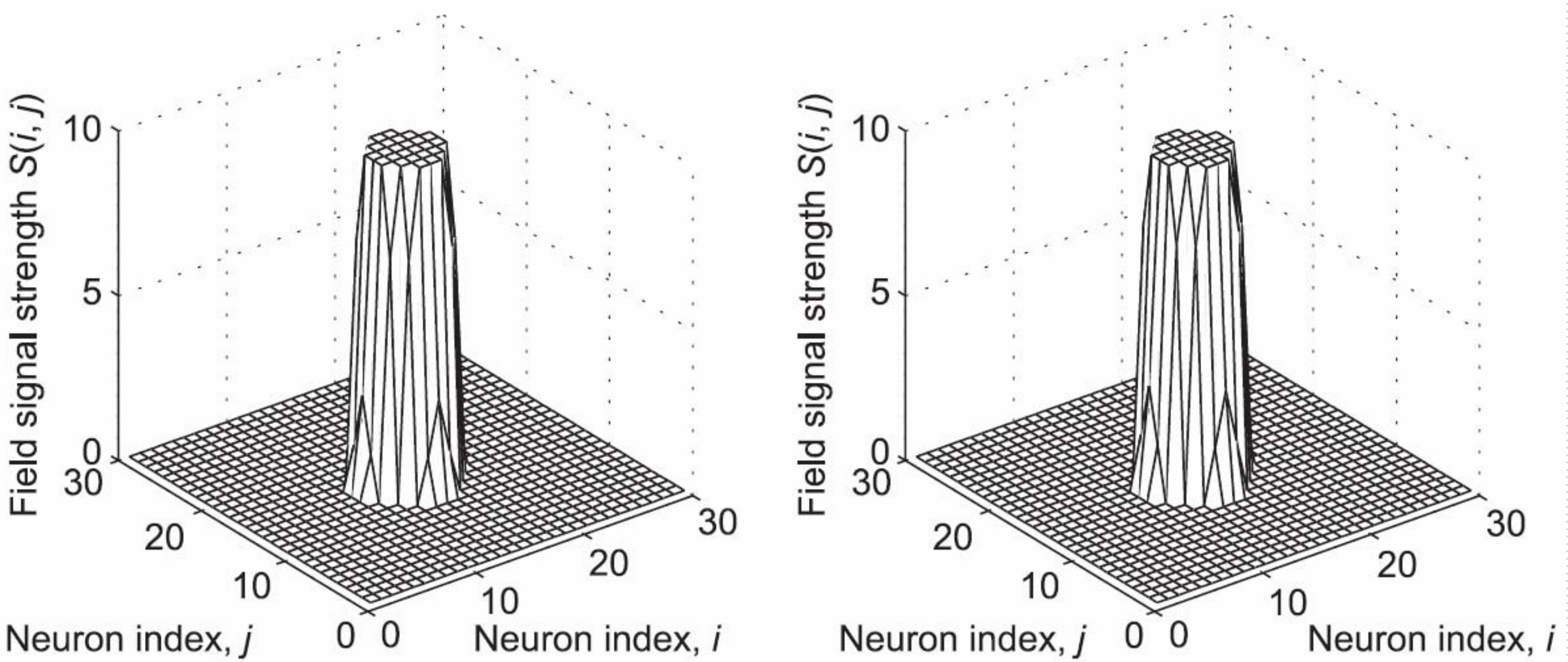
Two Dimensional Mexican Hat Network Simulation



Two Dimensional Mexican Hat Network Simulation



Two Dimensional Mexican Hat Network Simulation



Iteration 5

Iteration 6

Self-Organizing Feature Maps

- Dimensionality reduction + preservation of topological information common in normal human subconscious information processing
- Humans
  - routinely compress information by extracting relevant facts
  - develop reduced representations of impinging information while retaining essential knowledge
- Example: Biological vision
  - Three dimensional visual images routinely mapped onto a two dimensional retina
  - Information preserved to permit perfect visualization of a three dimensional world

Purpose of Intelligent Information Processing (Kohonen)

- Lies in the creation of simplified internal representations of the external world at different levels of abstraction

Computational Maps

- Early evidence for computational maps comes from the studies of Hubel and Wiesel on the primary visual cortex of cats and monkeys
- Specialized sensory areas of the cortex respond to the available spectrum of real world signals in an ordered fashion
- Example:
  - Tonotonic map in the auditory cortex is perfectly ordered according to frequency

A Hierarchy of Maps

Sequence of temporal processing

Primary map

Secondary map

Tertiary map

retains fine grained topological ordering as present in the original sensory signals

Topology Preservation

- Kohonen
  - “ . . . it will be intriguing to learn that an almost optimal spatial order, in relation to signal statistics can be completely determined in simple self-organizing processes under control of received information”

Topological Maps

- Topological maps preserve an order or a metric defined on the impinging inputs
- Motivated by the fact that representation of sensory information in the human brain has a geometrical order
- The same functional principle can be responsible for diverse (self-organized) representations of information—possibly even hierarchical

One Dimensional Topology Preserving Map

- m-neuron neural network
- i^{th} neuron produces a response s_i^k *in* response to input $I_k \in \Re^n$
- Input vectors $\{I_k\}$ are ordered according to some distance metric or in some topological way $I_1 \; R \; I_2 \; R \; I_3 \; . \; . \; .$, where R is some ordering relation

One Dimensional Topology Preserving Map

- Then the network produces a *one dimensional topology preserving map* if for $i_1 > i_2 > i_3$

$$s_{i_1}^1 = \max_j \{s_j^1\}$$

$$s_{i_2}^2 = \max_j \{s_j^2\}$$

$$s_{i_3}^3 = \max_j \{s_j^3\}$$

$$\vdots$$

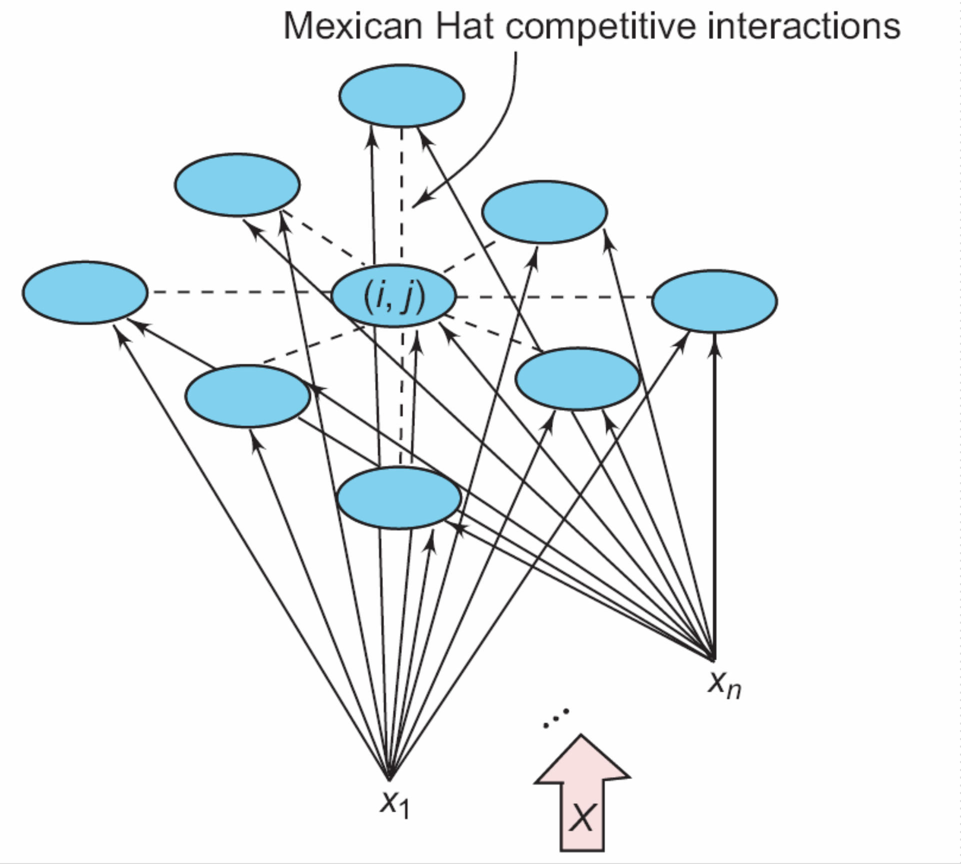
Self-Organizing Feature Map

- Finds its origin in the seminal work of von der Malsburg on self-organization
- Basic idea:
  - In addition to a genetically wired visual cortex there has to be some scope for self-organization of synapses of domain sensitive neurons to allow a local topographic ordering to develop

Self-Organizing Feature Map: Underlying Ideas

- Unsupervised learning process
- Is a competitive vector quantizer
- Real valued patterns are presented sequentially to a linear or planar array of neurons with Mexican hat interactions
- Clusters of neurons win the competition
- Weights of winning neurons are adjusted to bring about a better response to the current input
- Final weights specify clusters of network nodes that are topologically close
  - sensitive to clusters of inputs that are physically close in the input space
- Correspondence between signal features and response locations on the map
  - spatial location of a neuron in the array corresponds to a specific domain of inputs
- *Preserves the topology of the input*

SOFM Network Architecture

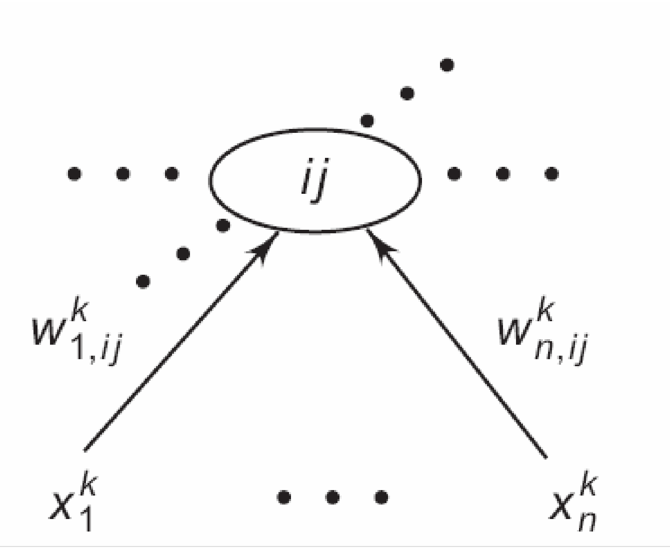


Requirements

- Distance relations in high dimensional spaces should be approximated by the network as the distances in the two dimensional neuronal field:
  - input neurons should be exposed to a sufficient number of different inputs
  - only the winning neuron and its neighbours adapt their connections
  - a similar weight update procedure is employed on neurons which comprise *topologically related subsets*
  - the resulting adjustment enhances the responses to the same or to a similar input that occurs subsequently

Notation

- Each neuron is identified by the double row-column index ij, $i, j = 1, \ldots, m$
- The ij^{th} neuron has an incoming weight vector $W_{ij}(k) = (w^k_{1,ij}, \ldots, w^k_{n,ij})$



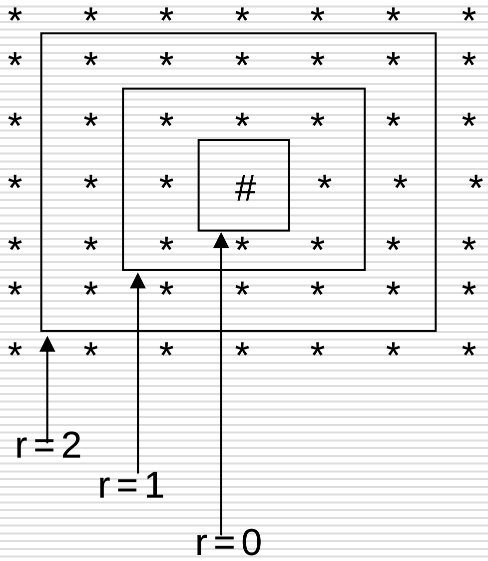
Neighbourhood Computation

- Identify a *neighbourhood* N_{IJ} around the winning neuron
- Winner identified by minimum Euclidean distance to input vector:

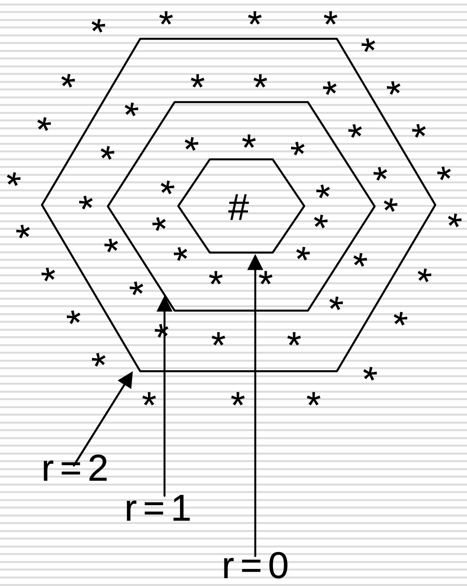
$$\|X_k - W_{IJ(k)}\| = \min_{i,j}\left\{\|X_k - W_{ij(k)}\|\right\}$$

- Neighbourhood is a function of time: as epochs of training elapse, the neighbourhood shrinks

Neighbourhood Shapes



Square neighbourhood



Hexagonal neighbourhood

Adaptation in SOFM

- Takes place according to the second generalized law of adaptation

$$\dot{w}_{l,ij} = \eta x_l s_{ij} - \gamma(s_{ij}) w_{l,ij}$$

- $\gamma\ (s_{ij}\ )$ may be chosen to be linear

$$\dot{w}_{l,ij} = \eta x_l s_{ij} - \beta s_{ij} w_{l,ij}$$

- Choosing $\eta = \beta$

$$\dot{w}_{l,ij} = \eta s_{ij}(x_l - w_{l,ij})$$

SOFM Adaptation

- Continuous time

$$\dot{W}_{ij} = \begin{cases} \eta(X - W_{ij}) & ij \in \mathcal{N}_{IJ} \\ 0 & ij \notin \mathcal{N}_{IJ} \end{cases}$$

- Discrete time

$$W_{ij(k+1)} = \begin{cases} W_{ij(k)} + \eta_k(X_k - W_{ij(k)}) & ij \in \mathcal{N}_{IJ}^k \\ W_{ij(k)} & ij \notin \mathcal{N}_{IJ}^k \end{cases}$$

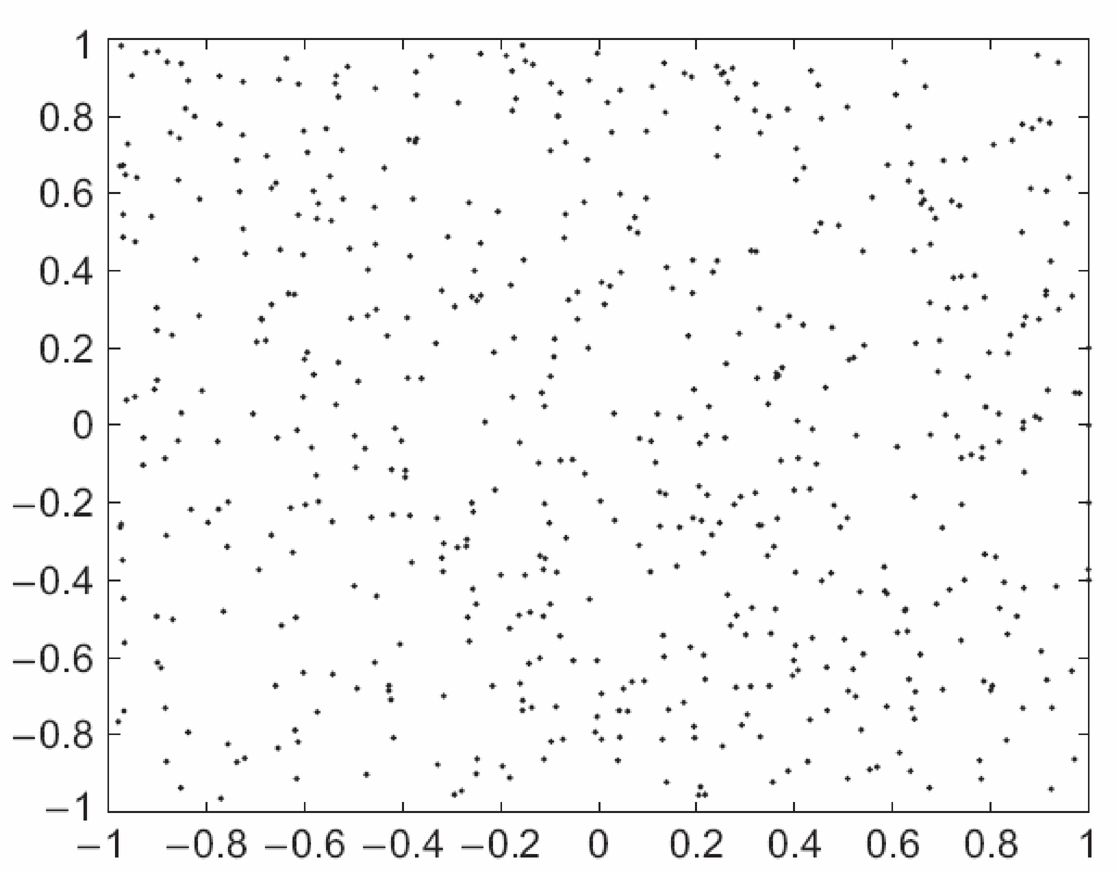
Some Observations

- *Ordering phase* (initial period of adaptation) *:* learning rate should be close to unity
- Learning rate should be decreased linearly, exponentially or inversely with iteration over the first 1000 epochs while maintaining its value above 0.1
- *Convergence phase:* learning rate should be maintained at around 0.01 for a large number of epochs
  - may typically run into many tens of thousands of epochs
- During the ordering phase N^k_{IJ} shrinks linearly with k to finally include only a few neurons
- During the convergence phase N^k_{IJ} may comprise only one or no neighbours

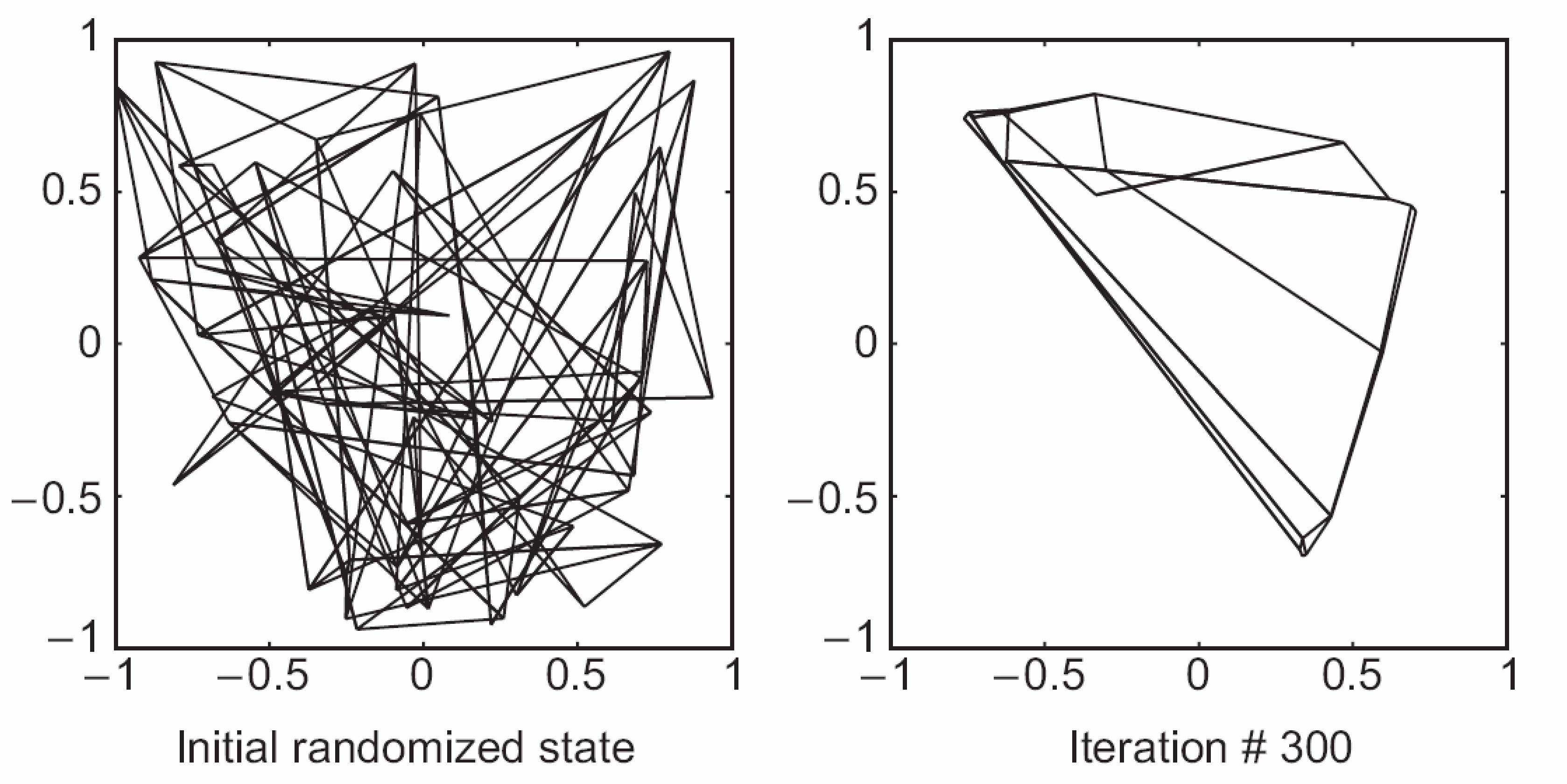
Simulation Example

The data employed in the experiment comprised 500 points distributed uniformly over the bipolar square $[-1, 1] \times [-1, 1]$

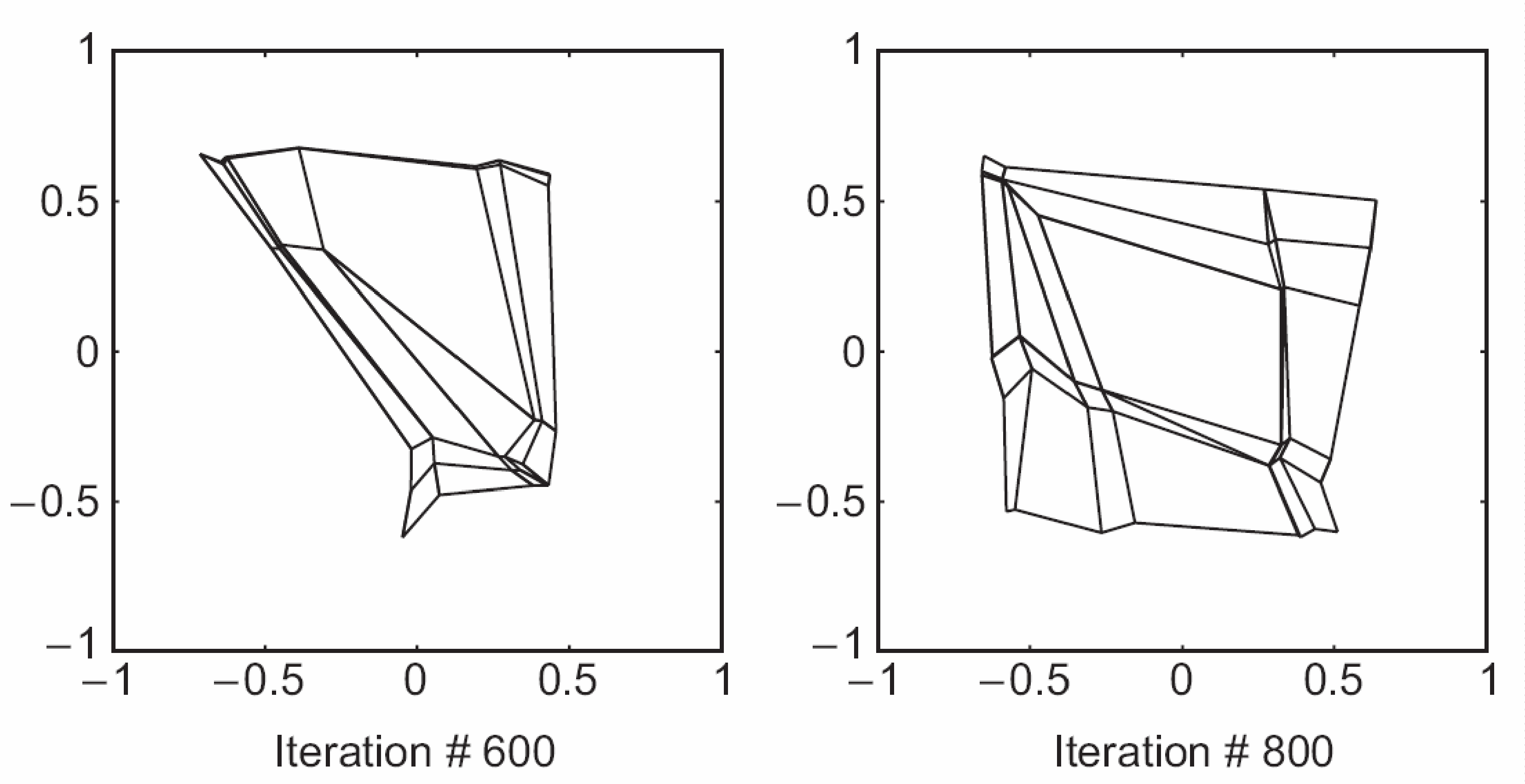
The points thus *describe* a geometrically square topology



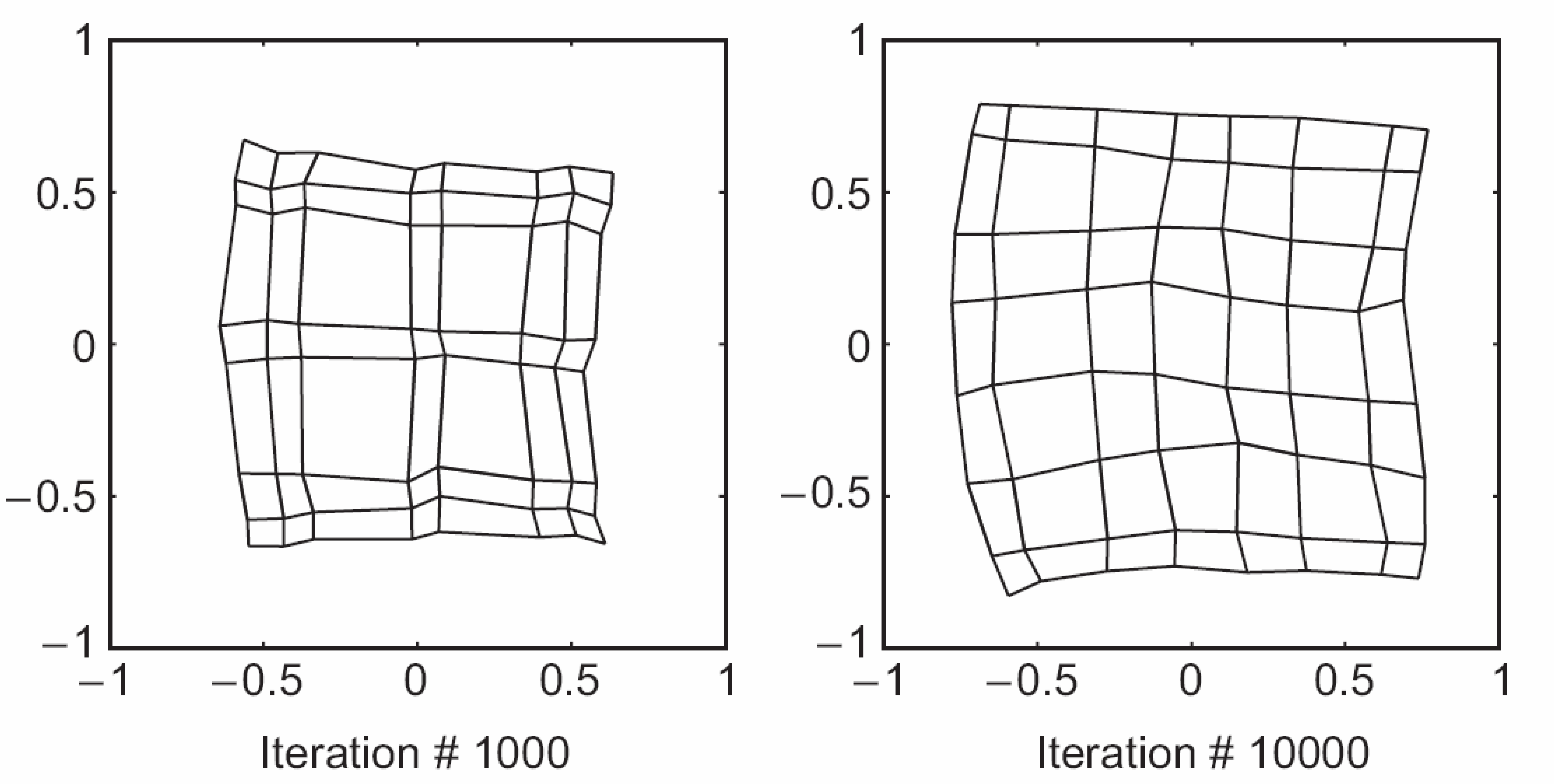
SOFM Simulation



SOFM Simulation



SOFM Simulation



Simulation Notes

- Initial value of the neighbourhood radius $r = 6$
  - Neighbourhood is initially a square of width 12 centered around the winning neuron IJ
- Neighbourhood width contracts by 1 every 200 epochs
- After 1000 epochs, neighbourhood radius maintained at 1
  - Means that the winning neuron and its four adjacent neurons are designated to update their weights on all subsequent iterations
  - Can also let this value go to zero which means that eventually, during the learning phase only the winning neuron updates its weights

MATLAB Code for SOFM

```
for epoch = 1:numpats*maxneuron
 count = count + 1;
 eta=0.9*(1 - epoch/1000);
  if (epoch > 999) eta = 0.005;
  end
 for p= 1:numpats
  for indx = 1:maxneuron
   for indy = 1:maxneuron
     dist(indx,indy)=sqrt((instarx(indx,
     indy)-data(1,p))^2 ...
                          +
     (instary(indx,indy)-data(2,p))^2);
   end
  end
[val1,rows]=min(dist);
 [val2,cols]=min(val1);
 indxmin=rows(cols);
 indymin=cols;
 for i=indxmin-nbd:indxmin+nbd
  for j=indymin-nbd:indymin+nbd
   if((i >=
    1)&(i<=maxneuron)&(j>=1)&(j<=max
    neuron))
    instarx(i,j)=instarx(i,j)+eta*(data(
    1,p)-instarx(i,j));
    instary(i,j)=instary(i,j)+eta*(data(
    2,p)-instary(i,j));
   end
  end
 end
end
```

Contd.

MATLAB Code for SOFM

```
for i=1:maxneuron
  plot(instarx(i,:),instary(i,:),'b.');
end
for i=1:maxneuron
  for j=1:maxneuron
    nb=[1 i-1 j
    2 i+1 j
    3 i j-1
    4 i j+1];
   for k=1:4

      if((nb(k,2)>=1)&(nb(k,2)<=maxneur
      on)&(nb(k,3)>=1)...

      &(nb(k,3)<=maxneuron))
line([instarx(i,j),instarx(nb(k,2),nb(k,3)
        )],...

        [instary(i,j),instary(nb(k,2),nb(k,3)
        )]);
      end
    end
  end
end
drawnow
if count == 200
  nbd = nbd - contractnbd;
if (nbd < 1) nbd = 1;
end
count = 0;
end
end
```

Operational Summary of the SOFM Algorithm

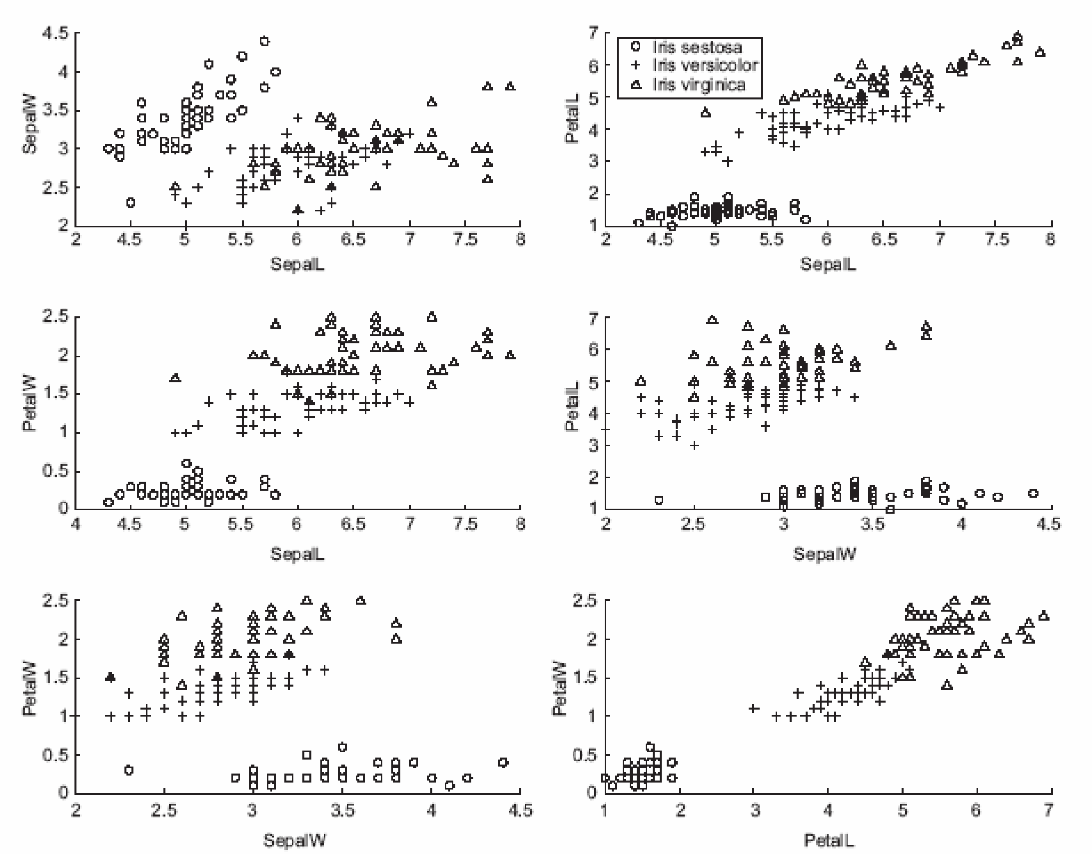
Given	A stream of training vectors $\{X_k\}_{k=1}^{Q}$ drawn uniformly from a possibly unknown probability distribution $p(X)$.
Initialize	$\looparrowright$ Weights $W_{ij(0)}$ to some small random numbers $\looparrowright$ Value of the neighbourhood $\mathcal{N}_{IJ}^{k}$ $\looparrowright$ Learning rate η_0

Iterate	Repeat
	{
	$\leadsto$ *Selection*: Pick a sample X_k
	$\leadsto$ *Similarity matching*: Find the winning neuron $(I\,J)$
	$\lVert X_k - W_{IJ(k)} \rVert = \min_{(1 \le i \le m)(1 \le j \le m)} \left\{ \lVert X_k - W_{ij(k)} \rVert \right\}$
	$\leadsto$ *Adaptation*: Update synaptic vectors of ONLY the winning cluster
	$w_{l,ij}^{k+1} = w_{l,ij}^{k} + \eta_k (x_l^k - w_{l,ij}^{k}) \quad ij \in \mathcal{N}_{IJ}^{k}$
	$\leadsto$ *Update*: Update $\eta_k, \mathcal{N}_{IJ}^{k}$
	
	} until (there is no observable change in the map)

Applications of the Self-organizing Map

- Vector quantization
- Neural phonetic typewriter
- Control of robot arms

Iris Pattern Classification



Iris Pattern Classification

Software on the Web

- Simulation performed with the SOFM MATLAB Toolbox available from www.cis.hut.fi/projects/somtoolbox
- Modified version of the program som demo2 used to generate the figures shown in this simulation.
- More applications, see text.