Introduction to Artificial Neural Networks

Lecture 10:

Self-Organizing Maps (SOM)

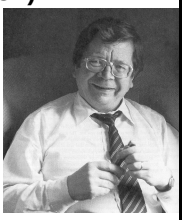
By: Ali Motie Nasrabadi

Outline

- Introduction
- SOM motivated by human brain
- Feature-mapping models
- Kohonen Model
- Self-Organizing Map
  - Competitive Process
  - Adaptive process
  - Feature Maps
- the concept of neighborhoods
- Computer Simulation
- Properties of the Feature Map
- Contextual Maps
- Application
- MATLAB TOOLBOX
- Clustering
- Vector Quantization
  - MATLAB TOOLBOX

Introduction

- **Self- Organizing Feature Maps (SOFM) also known as Kohonen maps or topographic maps were first introduced by von der Malsburg (1973) and in its present form by Kohonen (1982).**
- **According to Kohonen the idea of feature map formation can be stated as follows:**
  - **The** spatial location **of an output neuron in the topographic map corresponds to a particular domain, or feature of the input data.**



- **The output lattice characterizes a relative position of neurons with regards to its neighbors, that is their topological properties rather than exact geometric locations.**
- **In practice, dimensionality of the feature space is often restricted by its visualization aspect and typically is *l* = 1, 2 or 3.**

Introduction

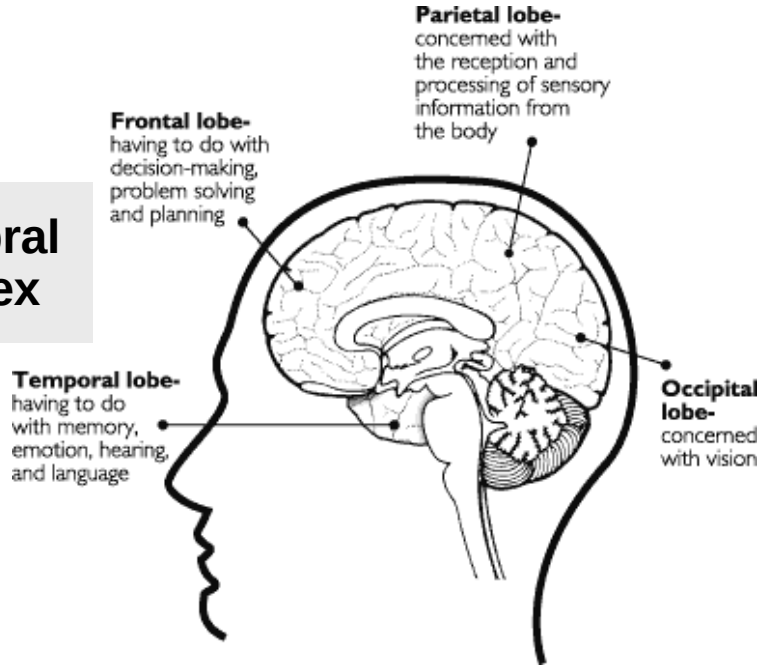
- **Based on competitive learning**
  - **one neuron per group is fired at any one time**
    - **winner-takes-all, winning neuron**
  - **winner-takes-all by lateral inhibitory connection**
- **Self-Organizing Map**
  - **neurons placed at the nodes of lattice**
    - **one or two dimension**
  - **neurons become selectively tuned to input patterns(stimuli)**
  - **by a competitive learning process**
  - **locations of neuron so tuned to be ordered**
    - **formation of topographic map of input pattern**
- **SOM is non-linear generalization of PCA**

SOM motivated by human brain

- **Brain is organized such a way that different sensory data is represented by topologically ordered computational maps**
  - **tactile, visual, acoustic sensory input are mapped onto areas of cerebral cortex in topologically ordered manner**
  - **building block of information processing infrastructure of neuvous system**
- **Neurons transform input signals into a place-coded probability distribution**
  - **sites of maximum relative activities within the map**
  - **accessed by higher-order processors with simple connection**
  - **each incoming information is kept in its proper context**
  - **Neurons dealing closed related information are close together so that they connected via short connections**

Lecture 10-5

Cerebral Cortex



http://www.health.org/pubs/qdocs/mom/TG/intro.htm

Lecture 10-6

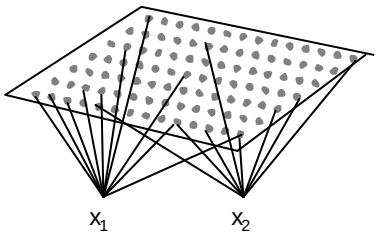
Feature-mapping models

- **Principle of topographic map formation**
  - **spatial location of output neuron in a TM corresponds to a domain or feature of data drawn from input space**
- **Willshaw-von der Malsburg Model**
  - **two 2D lattices : one for input neurons, others for output neurons**
  - **Hebbian type modifiable synapses**
  - **Postsynaptic lattice**
    - **short-range : excitatory**
    - **long-range : inhibitory**
  - **geometric proximity of presynaptic neurons is coded in the form of correlation, and it is used in postsynaptic lattice**
  - **specialized for mapping for same dimension of input and output**

Lecture 10-7

Kohonen Model

- **Captures essential features of computational maps in Brain**
- **More general and more attention than Willshaw-Malsburg model**
  - **capable of dimensionality reduction**
- **class of vector coding algorithm**
  - **optimally map into fixed number of code words**



Lecture 10-8

Self-Organizing Map

- Transform incoming signal pattern into discrete map
  - of 1-D or 2-D
  - adaptively topologically ordered fashion
- input pattern is represented as a localized region or spot of activities in the network
- After initialization, three essential processes
  - competition
    - largest discriminant function selected
    - winner of competition
  - cooperation
    - spatial neighbors of winning neuron is selected
  - synaptic adaptation
    - excited neurons adjust synaptic weights

Competitive Process

- Find best match of input vector with synaptic weight

  $x = [x_1, x_2, \ldots, x_3]^T$

  $w_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T$, $j = 1, 2, 3, l$

- Best matching, winning neuron

  $i(x) = \arg\min \|x - w_j\|$, $j = 1,2,3,..,l$

- continuous input space is mapped onto discrete output space of neuron by competitive process

Cooperative Process

- For a winning neuron, immediate neighborhood excite more than farther away
- topological neighborhood decay smoothly with lateral distance
  - symmetric about maximum point defined by $d_{ij} = 0$
  - monotonically decreasing to zero for $d_{ij} \rightarrow 0$
  - neighborhood function: Gaussian case
  - 2-D lattice $d_{j,i}^2 = \|r_j - r_i\|^2$

$$h_{j,i(x)} = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right)$$

- Size of neighborhood shrinks with time

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right), \quad n = 0,1,2,3$$

Lecture 10-11

Adaptive process

- Synaptic weight vector is changed in relation with input vector

$$w_j(n+1) = w_j(n) + \eta(n)\, h_{j,i(x)}(n)\,(x - w_j(n))$$

- applied to all neurons inside the neighborhood of winning neuron ***i***
- effect of moving weight w_j toward input vector x
- upon repeated presentation of the training data, weight tend to follow the distribution
- Learning rate $\eta(n)$: decay with time
- may decompose two phases
  - self-organizing phase : with shrinking neighborhood radius
  - convergence phase : after ordering, for good statistical accuracy

Lecture 10-12

Summary of SOM

- **Continuous input space governed by underlying probability distribution**
- **lattice of network defines a discrete output space**
- **time-varying neighborhood function defined around winning neuron**
- **learning rate decrease gradually with time**

1.initialize w's

2. find nearest cell

$$i(\underline{x}) = \text{argmin}_j \| \underline{x}(n) - \underline{w}_j(n) \|$$

3. update weights of neighbors

$$\underline{w}_j(n+1) = \underline{w}_j(n) + \eta(n)\, h_{j,i(x)}(n) [\, \underline{x}(n) - \underline{w}_j(n) \,]$$

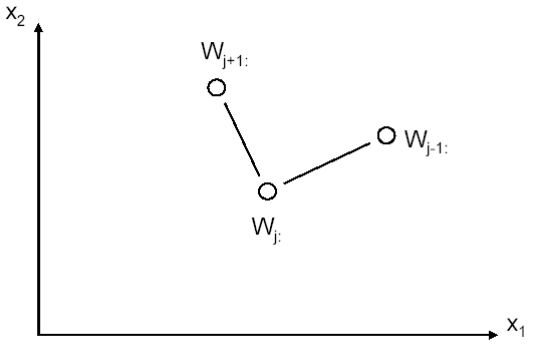
4. reduce neighbors and η

5. Go to 2

Lecture 10-13

Feature Maps

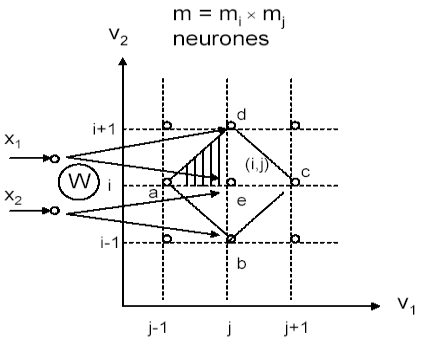
- **A** Feature Map **is a plot of synaptic** weights **in the** input space **in which weights of the neighboring neurons are joined by lines or plane segments (patches).**
- **The feature map has the following structure:**



A feature map for a (2-D, 1-D) SOFM

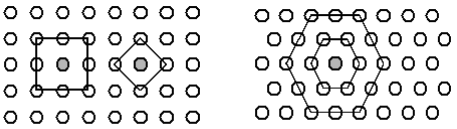
Lecture 10-14

- **Note that in the feature map the point representing the weight vector,** w***j*** **, is joined by line segments with points representing weights** w***j*** **-1 and** w***j*** **+1 because neurons *j* - 1, *j*, and *j* + 1 are located in the adjacent positions of the 1-D lattice.**
- Example: 2-D input space, 2-D feature space **Let us consider a SOFM with two inputs (*p* = 2) and *m* neurons arranged in a 2-D lattice as in Figure 8–5.**

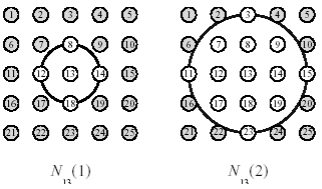


The concept of neighborhoods

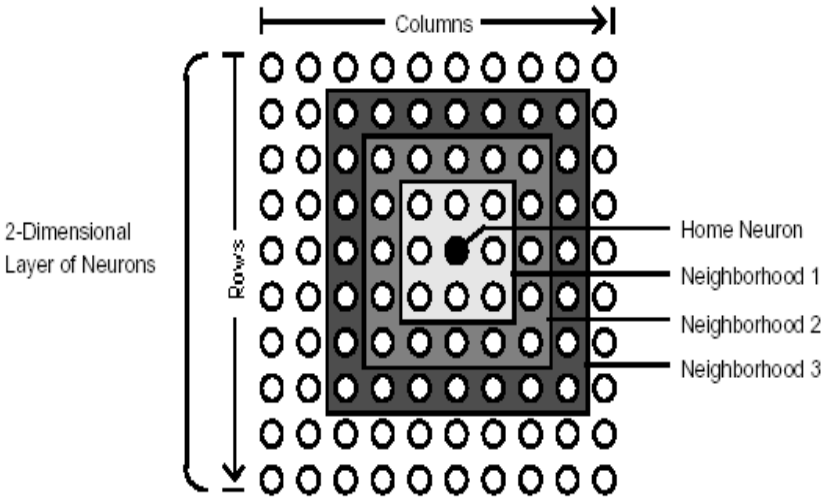
- **The different type of neighborhood**



- **The left diagram shows a two-dimensional neighborhoo of radius d=1 around neuron . The right diagram shows a neighborhood of radius d=2.**

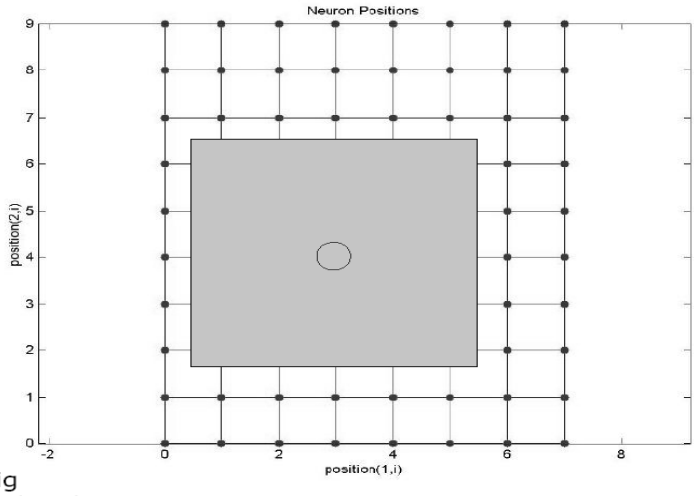


Reducing the Neighborhood (1)



Lecture 10-17

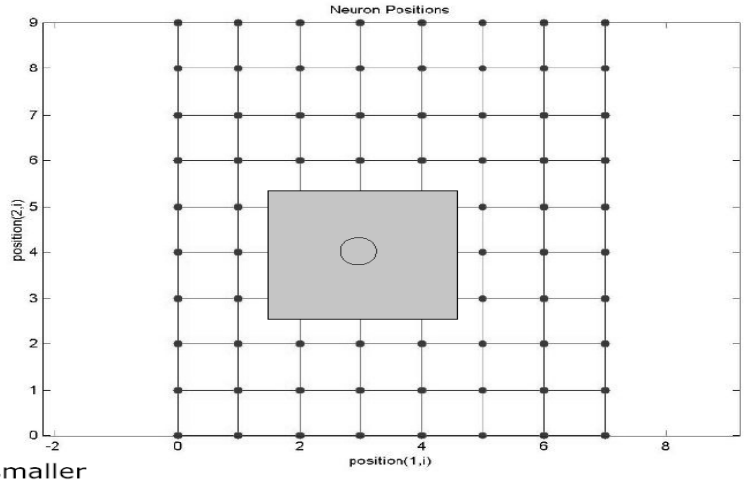
Reducing the Neighborhood (2)



First, big neighborhood

Lecture 10-18

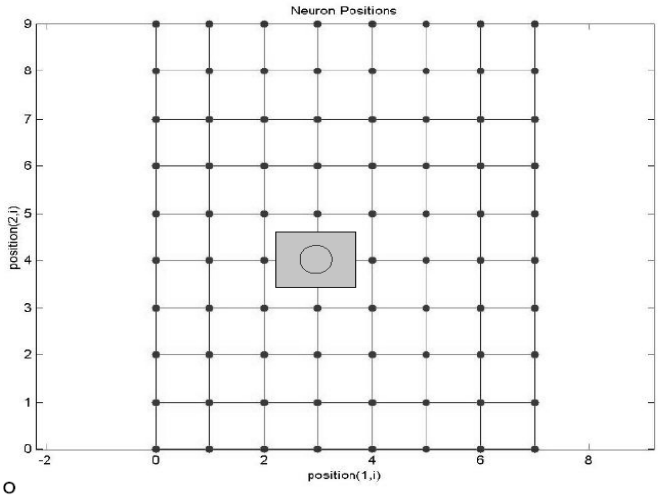
Reducing the Neighborhood (3)



Then, smaller neighborhood

Lecture 10-19

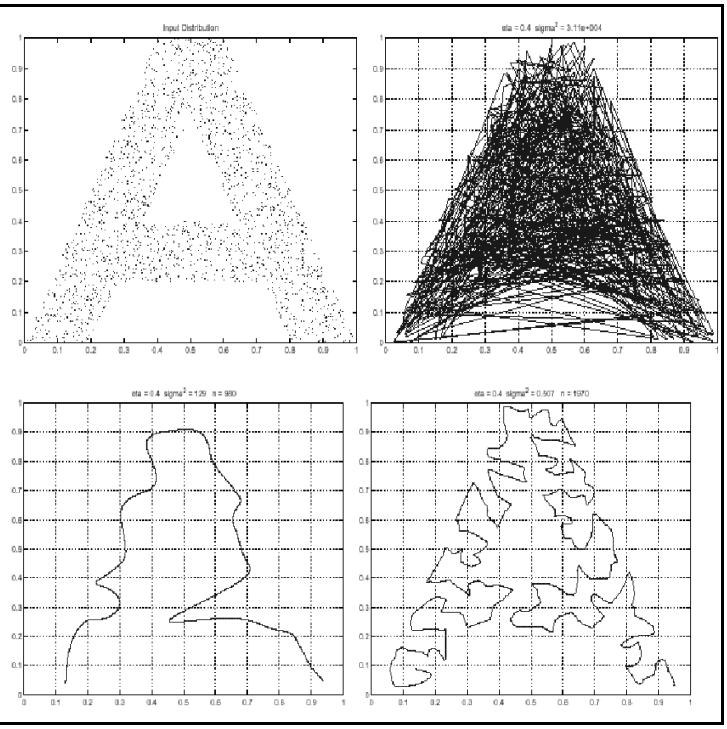
Reducing the Neighborhood (4)



Then, no neighborhood

Lecture 10-20

■ **Demo of one dim. map**



Computer Simulation

- **Input sample : random numbers within 2-D unit square**
- **100 neurons (10x10)**
- **Initial weights : random assignment (0.0~1.0)**
- **Display**
  - **each neuron positioned at w_1, w_2**
  - **neighbors are connected by line**

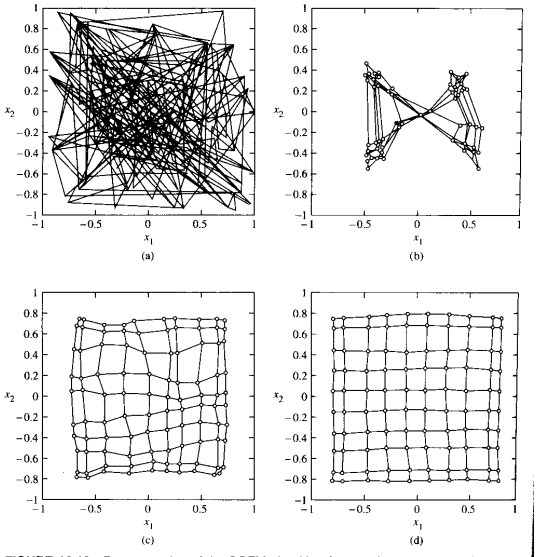


FIGURE 10.12 Demonstration of the SOFM algorithm for a uniformly distributed two-dimensional input, using a two-dimensional lattice. (a) Initial random weights. (b) Network after 50 iterations. (c) After 1000 iterations. (d) After 10,000 iterations.

Lecture 10-23

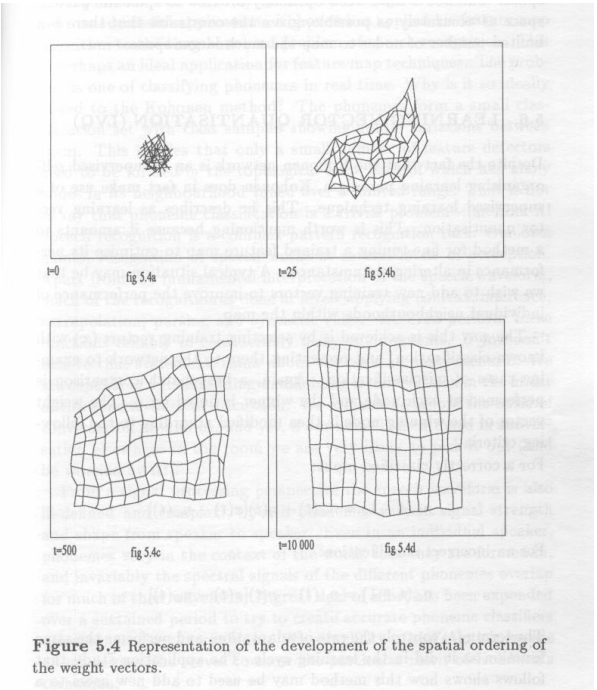
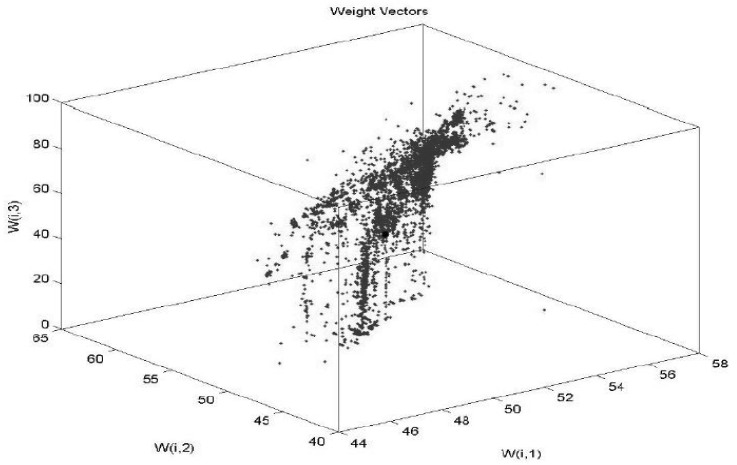


Figure 5.4 Representation of the development of the spatial ordering of the weight vectors.

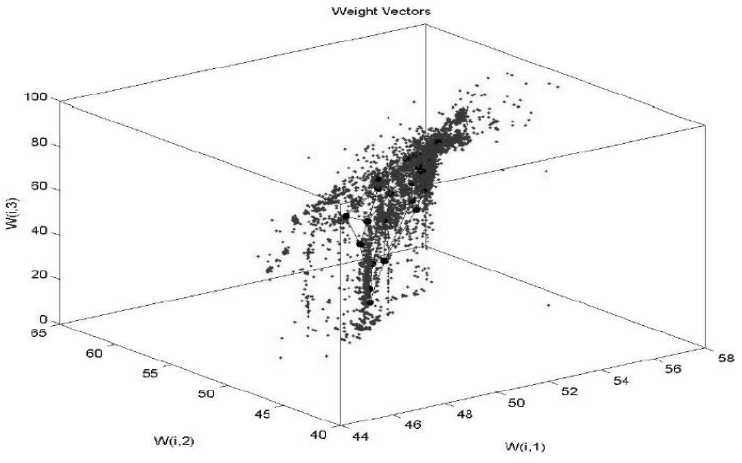
Lecture 10-24

Example: 3D input space

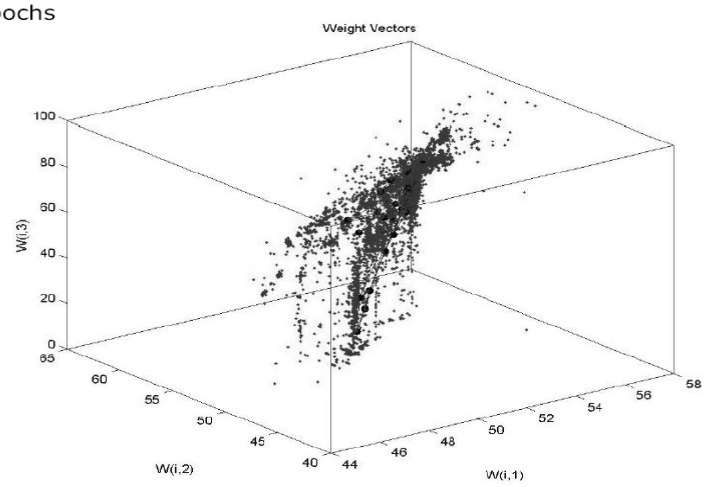
Initial



5 epochs

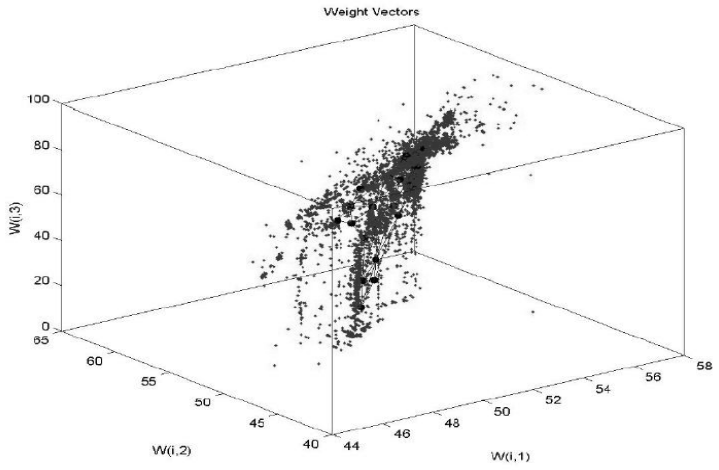


10 epochs



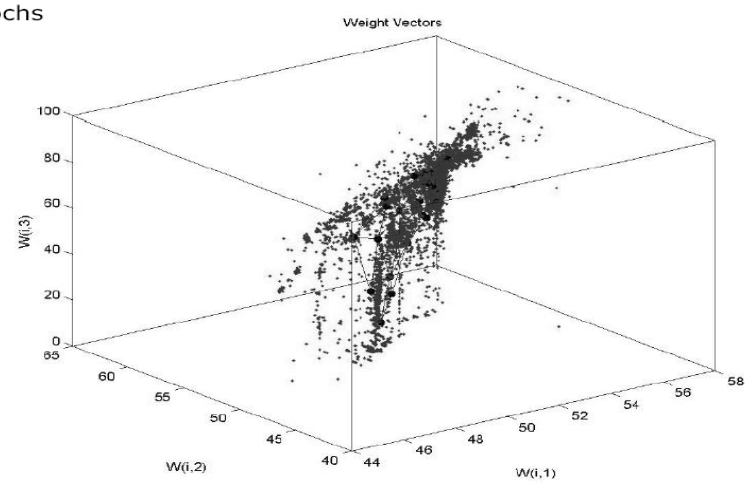
Lecture 10-27

15 epochs



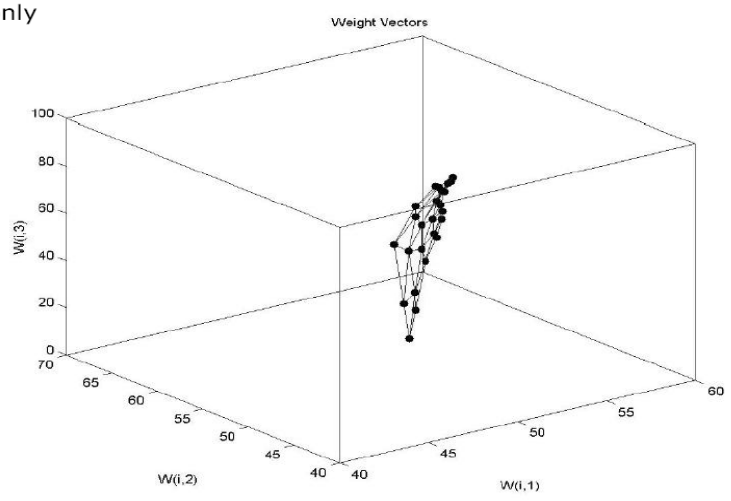
Lecture 10-28

20 epochs

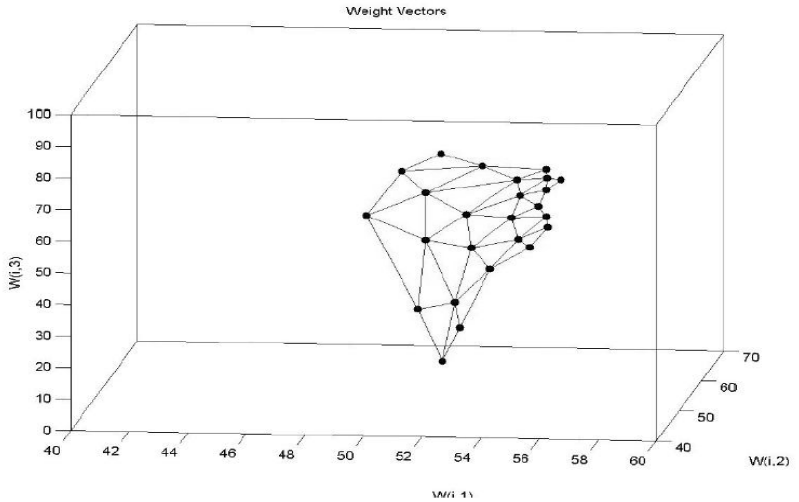


Lecture 10-29

SOM only



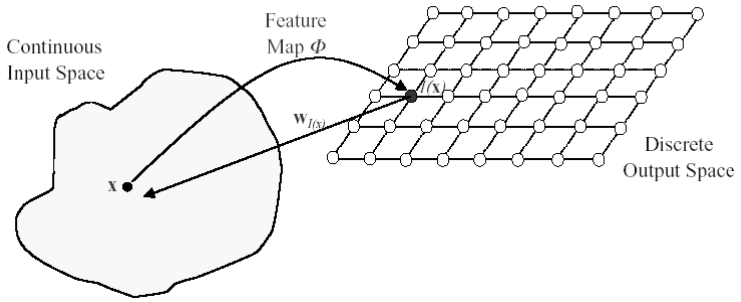
Lecture 10-30



Lecture 10-31

Properties of the Feature Map (1)

- **Once the SOM algorithm has converged, the feature map displays important statistical characteristics of the input space. Given an input vector** x**, the feature map F provides a winning neuron** ***I*****(**x**) in the output space, and the weight vector** w***I*****(**x**) provides the coordinates of the image of that neuron in the input space.**



10-32

Properties of the Feature Map (2)

- **The feature map has four important properties that we shall look at in turn:**
  - Property 1 : Approximation of the Input Space
  - Property 2 : Topological Ordering
  - Property 3 : Density Matching
  - Property 4 : Feature Selection

Property 1 : Approximation of the Input Space

- *The feature map* **F** *represented by the set of weight vectors* {wi} *in the output space, provides a good approximation to the input space.*
  - **We can state the aim of the SOM as storing a large set of input vectors {**x**} by finding a smaller set of prototypes {**w***i***} so as to provide a good approximation to the original input space. The theoretical basis of this idea is rooted in** *vector quantization theory***, the motivation of which is dimensionality reduction or data compression.**
  - **In effect, the goodness of the approximation is given by the total squared distance**

$$D = \sum_{\mathbf{x}} \left\| \mathbf{x} - \mathbf{w}_{I(\mathbf{x})} \right\|^2$$

  - **which we wish to minimize. If we work through gradient descent style mathematics we do end up with the SOM weight update algorithm, which confirms that it is generating a good approximation to the input space.**

Property 2 : Topological Ordering

- *The feature map* **F** *computed by the SOM algorithm is topologically ordered in the sense that the spatial location of a neuron in the output lattice/grid corresponds to a particular domain or feature of the input patterns.*
  - **The topological ordering property is a direct consequence of the weight update equation that forces the weight vector** $\mathbf{w}_{i(\mathbf{x})}$ **of the winning neuron** $i(\mathbf{x})$ **to move toward the input vector** $\mathbf{x}$**. The crucial factor is that the weight updates also move the weight vectors** $\mathbf{w}_j$ **of the closest neighboring neurons** j **along with the winning neuron** $i(\mathbf{x})$**. Together these weight changes cause the whole output space to become appropriately ordered.**
  - **We can visualize the feature map F as an** *elastic or virtual net* **with a grid like topology. Each output node can be represented in the input space at coordinates given by their weights. Then if the neighbouring nodes in output space have their corresponding points in input space connected together, the resulting image of the output grid reveals directly the topological ordering at each stage of the network training.**

Property 3 : Density Matching

- *The feature map* **F** *reflects variations in the statistics of the input distribution: regions in the input space from which the sample training vectors* $\mathbf{x}$ *are drawn with high probability of occurrence are mapped onto larger domains of the output space, and therefore with better resolution than regions of input space from which training vectors are drawn with low probability.*
  - **We need to relate the input vector probability distribution** $p(\mathbf{x})$ **to the** ***magnification factor*** $m(\mathbf{x})$ **of the feature map. Generally, for two dimensional feature maps the relation cannot be expressed as a simple function, but in one dimension we can show that**

$$m(\mathbf{x}) \propto p^{2/3}(\mathbf{x})$$

  - **So the SOM algorithm doesn't match the input density exactly, because of the power of 2/3 rather than 1. Computer simulations indicate similar approximate density matching in general, always with the low input density regions slightly over-represented.**

Property 4 : Feature Selection

- *Given data from an input space with a non-linear distribution, the self organizing map is able to select a set of best features for approximating the underlying distribution.*
  - **This property is a natural culmination of properties 1 through 3.**
  - **Remember how Principal Component Analysis (PCA) is able to compute the input dimensions which carry the most variance in the training data. It does this by computing the eigenvector associated with the largest eigenvalue of the correlation matrix.**
  - **PCA is fine if the data really does form a line or plane in input space, but if the data forms a curved line or surface (e.g. a semi-circle), linear PCA is no good, but a SOM will overcome the approximation problem by virtue of its topological ordering property.**
  - **The SOM provides a discrete approximation of finding so-called** *principal curves* **or** *principal surfaces***, and may therefore be viewed as a non-linear generalization of PCA.**

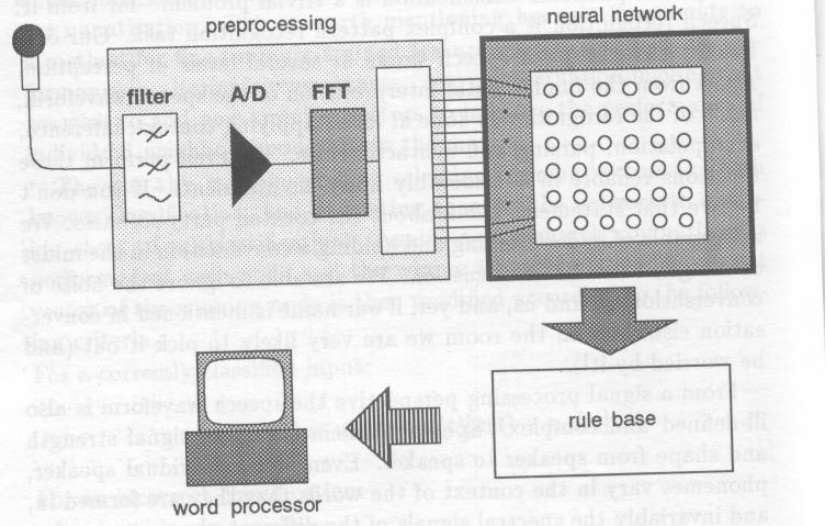
Contextual Maps

- **Visualization of SOM**
  - **elastic net with synaptic weight vectors treated as pointers**
  - **class labels are assigned to neurons in a 2-D lattice called contextual map**
- **Contextual map**
  - **lattice is partitioned into coherent regions**
    - **trained by $[x_s, x_a]^T$ where x_s is symbol code, and x_a is attribute code**
    - **test pattern defined by $[x_s, 0]^T$**
    - **strongest response identified**
  - **called contextual map or semantic map**
    - **resembles cortical map (maps formed in cerebral cortex)**

Application: The Phonetic Typewriter

- **Part of a Finnish speech recognition system**
- **Sampled at 13kHz**
- **Passed through low pass filter (5.3 kHz), 12 bit A/D, 256 point FFT,**
- **sliced into 9.83mS time windows**
- **Network input: 16-bit feature vector consisting of power spectrum of 9.83 mS time window**
- Note: **phonemes are much longer, 40 – 400 mS.**

- points about phonemic typewriter
  - **Network "discovers" phonemes as important features (Kohonen claims).**
  - **Figure shows nodes labeled subjectively by hand after training.**
  - **Due to the topographic nature of the map, similar sounding phonemes cause nearby nodes to fire.**
  - **If used for word recognition, errors will therefore produce similar**
  - **sounding words to the original word.**



ure 10-41

a	a	a	ah	h	æ	æ	ø	ø	e	e	e
o	a	a	h	r	æ	l	ø	y	y	j	i
o	o	a	h	r	r	r	g	g	y	j	i
o	o	m	a	r	m	n	m	n	j	i	i
l	o	u	h	v	vm	n	n	h	hj	j	j
l	u	v	v	p	d	d	t	r	h	hi	j
.	.	u	v	tk	k	p	p	p	r	k	s
.	.	v	k	pt	t	p	t	p	h	s	s

Figure 5.6 A phoneme feature map. Kohonen calls this a "phonotopic map".

Lecture 10-42

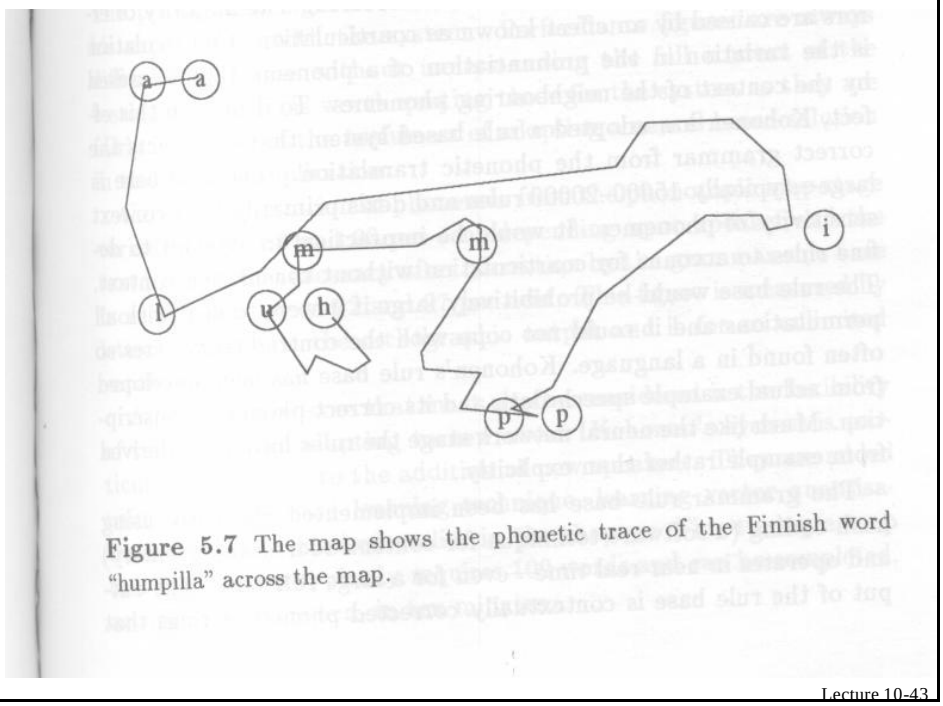


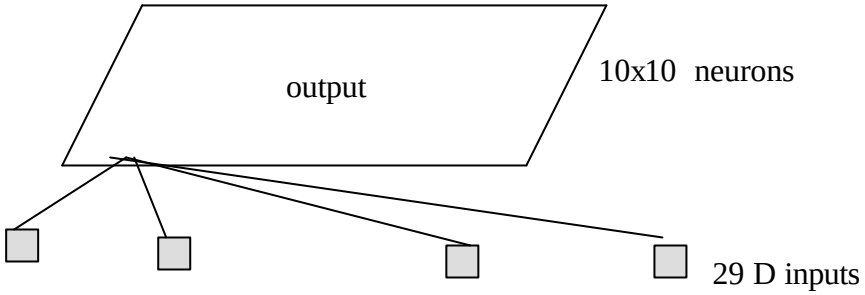
Figure 5.7 The map shows the phonetic trace of the Finnish word "humpilla" across the map.

Lecture 10-43

Example: Animal and attributes, 16 animals with 13 attributes

Animal	Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	cow
small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
is medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
big																
2 legs																
4 legs																
has hair																
hooves																
mane						see Haykin's book										
feathers																
hunt																
likes run																
to fly																
swim																

Lecture 10-44



lion=(0,0,0,0,0,0,0,0,0,0, **0.2**, 0,0,0, 0,0,1,0,1,1,0,1,0,1,1,0,0)

0,0,0,0,0,0,0,0,0,0, **0.2**, 0,0,0 =animal code

Train for 2000 interations of all animals (until convergence)

Next the code for each animal is presented but with the animal code all zeroes. We then assign the neuron to the animal for which it gives the largest response

Lecture 10-45

dog	dog	fox	fox	fox	cat	cat	cat	eagle	eagle
dog	dog	fox	fox	fox	cat	cat	cat	eagle	eagle
wolf	wolf	wolf	fox	cat	tiger	tiger	tiger	owl	owl
wolf	wolf	lion	lion	lion	tiger	tiger	tiger	hawk	hawk
wolf	wolf	lion	lion	lion	tiger	tiger	tiger	hawk	hawk
wolf	wolf	lion	lion	lion	owl	dove	hawk	dove	dove
horse	horse	lion	lion	lion	dove	hen	hen	dove	dove
horse	horse	zebra	cow	cow	cow	hen	hen	dove	dove
zebra	zebra	zebra	cow	cow	cow	hen	hen	duck	goose
zebra	zebra	zebra	cow	cow	cow	duck	duck	duck	goose

FIGURE 9.18 Semantic map obtained through the use of simulated electrode penetration mapping. The map is divided into three regions representing: birds, peaceful species, and hunters.

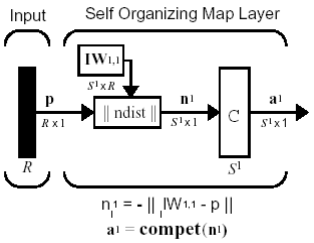
Lecture 10-46

MATLAB TOOLBOX

- net = newsom(PR,[d1,d2,...],tfcn,dfcn,olr,osteps,tlr,tns)
  - Description of function
    - Competitive layers are used to solve classification problems.
  - NET = NEWSOM(PR,[D1,D2,...],TFCN,DFCN,OLR,OSTEPS,TLR,TNS) takes,
    - PR - Rx2 matrix of min and max values for R input elements.
    - Di - Size of ith layer dimension, defaults = [5 8].
    - TFCN - Topology function, default = 'hextop'.
    - DFCN - Distance function, default = 'linkdist'.
    - OLR - Ordering phase learning rate, default = 0.9.
    - OSTEPS - Ordering phase steps, default = 1000.
    - TLR - Tuning phase learning rate, default = 0.02;
    - TND - Tuning phase neighborhood distance, default = 1.

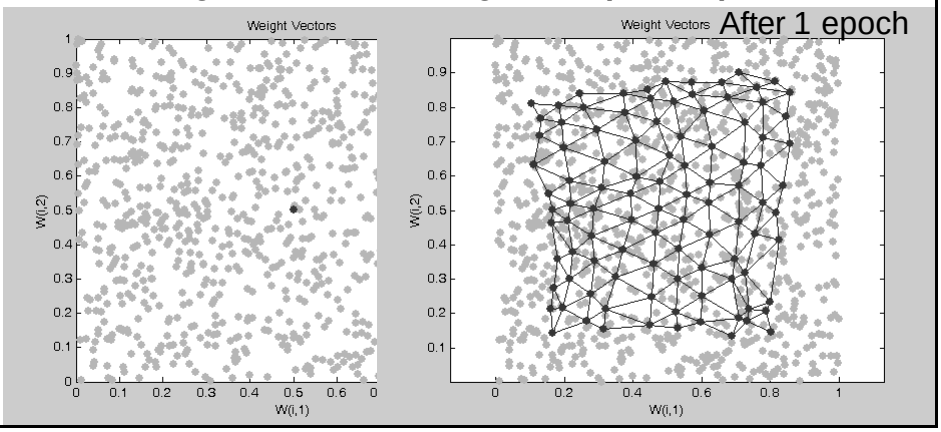
    and returns a new self-organizing map.

- The architecture for this SOFM is shown below



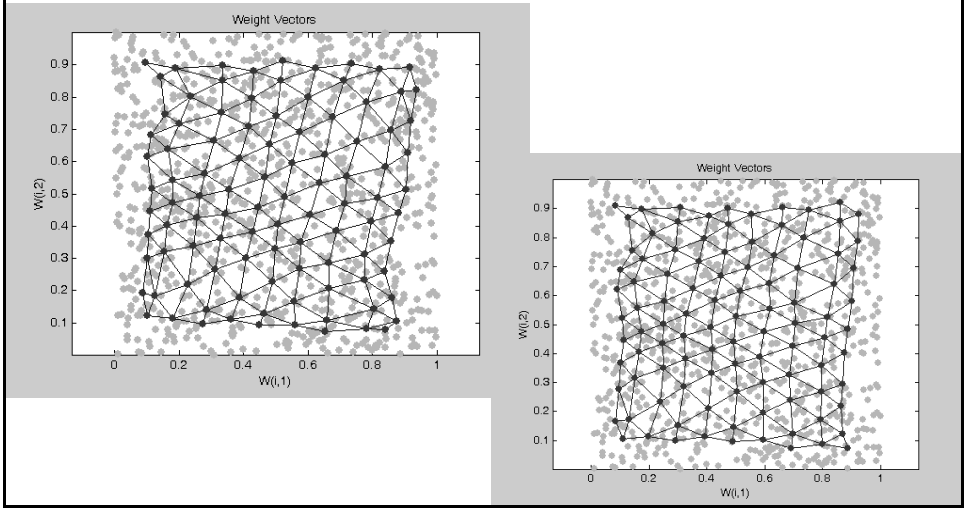
example

- **Input sample : random numbers within 2-D unit square**
- **100 neurons (10x10)**
- **Initial weights : random assignment (0.0~.01)**



After 1 epoch

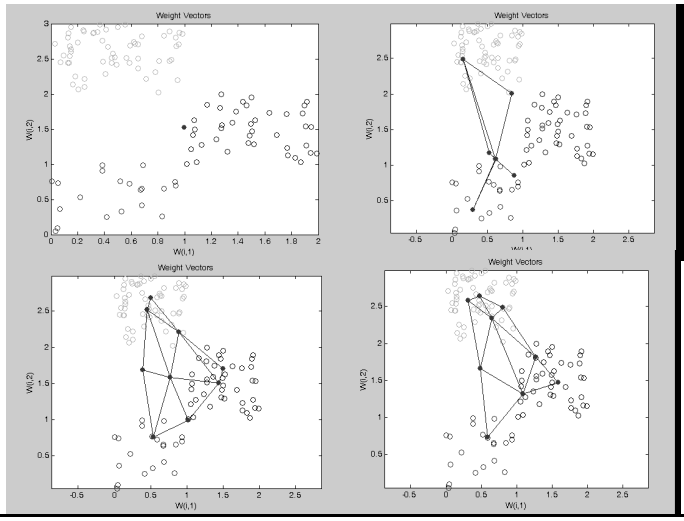
- **After 26 epoch**



Example 2: the 2_D Pattern with 3 clusters of Data

- **N1=20, N2= 40, N3= 60, som:[3 3] Initial Random weights**

Initial Random weights

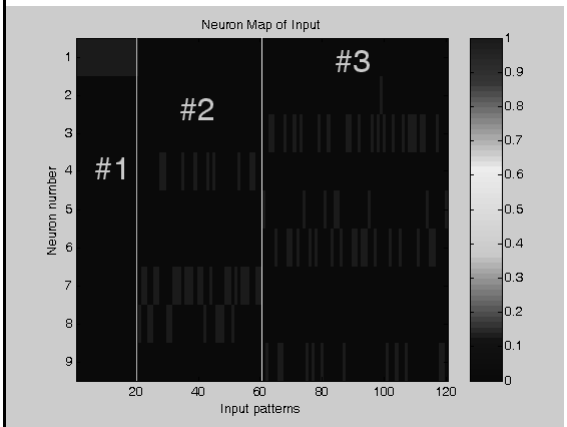


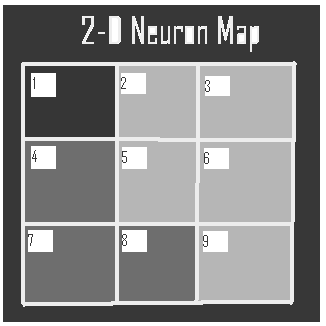
Network After 1 Epoch

Network After 25 Epochs

Network After 100 Epochs

- **The neuron map**





■ **M_file of SOM example**

```
%classification with SOM
%inputs are random varaible
rand('seed',0)
P1=rand(20,2)';
P2=1+rand(40,2)';
P3=[0*ones(60,1) 2*ones(60,1)]'+rand(60,2)';
% training data
PL=[P1 P2 P3];
plot(P1(1,:),P1(2,:),'or',P2(1,:),P2(2,:),'ob',P3(1,:),P3(2,:),'og')
xlabel('x1');ylabel('x2');title('Scatter plot of data')
net = newsom(minmax(PL),[3 3]);
hold on;plotsom(net.iw{1,1},net.layers{1}.distances)
net.trainParam.epochs = 1;
net = train(net,PL);
figure;
plot(P1(1,:),P1(2,:),'or',P2(1,:),P2(2,:),'ob',P3(1,:),P3(2,:),'og')
xlabel('x1');ylabel('x2');title('Scatter plot of data')
hold on;plotsom(net.iw{1,1},net.layers{1}.distances)
%plot the weights after 25 epochs
net.trainParam.epochs = 24;
net = train(net,PL);
figure;
plot(P1(1,:),P1(2,:),'or',P2(1,:),P2(2,:),'ob',P3(1,:),P3(2,:),'og')
xlabel('x1');ylabel('x2');title('Scatter plot of data')
hold on;plotsom(net.iw{1,1},net.layers{1}.distances)
%plot the weights after 100 epochs
net.trainParam.epochs = 75;
net = train(net,PL);
figure;
plot(P1(1,:),P1(2,:),'or',P2(1,:),P2(2,:),'ob',P3(1,:),P3(2,:),'og')
xlabel('x1');ylabel('x2');title('Scatter plot of data')
hold on;plotsom(net.iw{1,1},net.layers{1}.distances)

%calculation of learning error
Y = sim(net,PL)
Yc = vec2ind(Y);
rYc=fix(Yc/3);
cYc=mod(Yc,3);
```


Clustering

- **Group ‘similar’ ones together**
- **Similarity function : s(x, x’)**
  - **angle between two vectors**

$$s(\mathbf{x},\mathbf{x}') = \frac{\mathbf{x}^t\mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}$$

$$s(\mathbf{x},\mathbf{x}') = \frac{\mathbf{x}^t\mathbf{x}'}{d}$$

$$s(\mathbf{x},\mathbf{x}') = \frac{\mathbf{x}^t\mathbf{x}'}{\mathbf{x}^t\mathbf{x}+\mathbf{x}'^t\mathbf{x}'-\mathbf{x}^t\mathbf{x}'}$$

Clustering quality measure

- **sum-of-square errors**

$$J = \sum_{c} \sum_{x \in X} \| x - m_c \|^2$$

- **Related minimum variance**
  - **where**

$$J = \frac{1}{2} \sum_{i=1}^{c} n_i \bar{s}_i$$

$$\bar{s}_i = \frac{1}{n_i^2} \sum_{x \in \aleph_i} \sum_{x' \in \aleph_i} \| x - x' \|$$

- **can be average, median, maximum distance between points**

Lecture 10-55

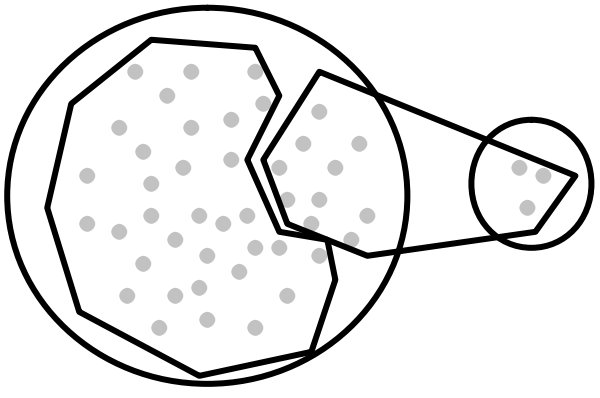
Iterative Optimization

- **Basic idea**
  - **if moving x from C_i to C_j improves J, then move.**
- **Isodata algorithm**
  1. **select initial value of mean for each class**
  2. **classify n samples to nearest mean**
  3. **recompute means**
  4. **If any mean changed, go to 2**

Lecture 10-56

Renormalized Training Algorithm

- 1. Initialize
  - number of code vector to 2;
  - initialize position randomly from training set
- 2. selection of input vector randomly from training set
- 3. Encoding of the input vector
  - determine winning code vector; nearest neighbor
- 4. updating code book
  - usual winner and its neighbor
- 5. splitting code book
  - code book update(step 4) using a new training input
  - until updated 10-30 times of the number of code vectors
  - splitting by interpolation; take a half way
- 6. Go to 2 or completion of training



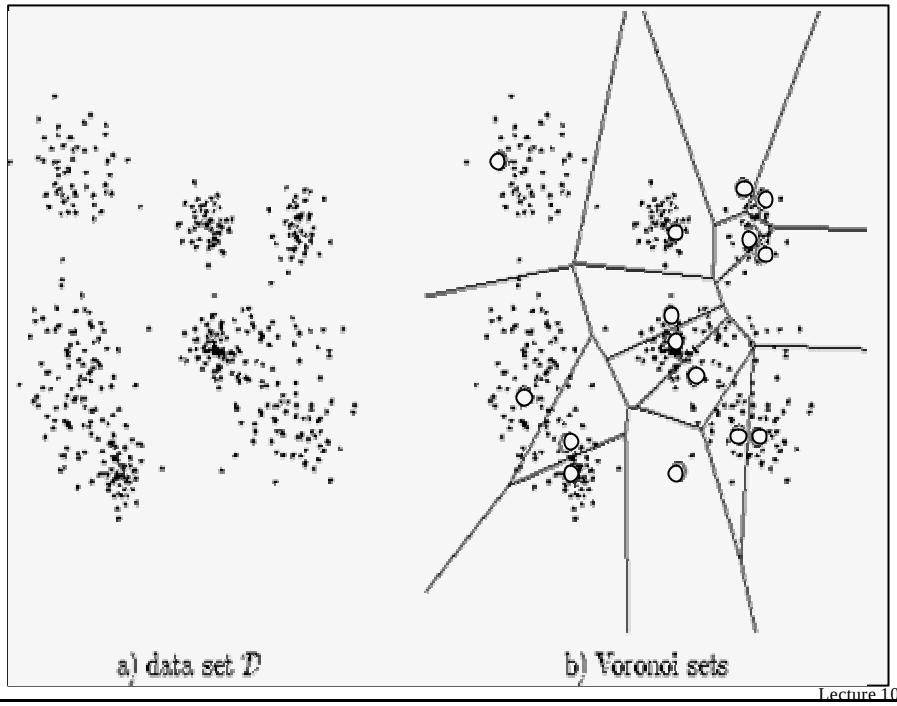
Hierarchical Clustering

- Sequence of partitions of N samples into C clusters
  - 1. N clusters
  - 2. Merge nearest two clusters to make total N-1 cluster
  - 3. do until the number of clusters C
- Dendrogram
  - Agglomerative : bottom up
  - divisive : top down
- Distance between sets
  - $d_{min}(X_i, X_j) = \min \| X - X' \|$
  - $d_{max}(X_i, X_j) = \max \| X - X' \|$
  - $d_{average}(X_i, X_j) =$
  - $d_{mean}(X_i, X_j) = \|m_i - m_j\|$

Lecture 10-59

- Categorize a given set of input vectors into M classes using competitive learning algorithms, and then represent any vector just by the class into which it falls
- Important use of competitive learning (esp. in data compressing) divides entire pattern space into a number of separate subspaces set of *M* units represent set of prototype vectors: CODEBOOK (cf k-means)
- new pattern x is assigned to a class based on its closeness to a prototype vector using Euclidean distances LABELED (cf k-nearest neighbours, kernel density estimation

Lecture 10-60



Vector Quantization

- **VQ : data compression technique**
  - **input space is divided into distinct regions**
    - **reproduction vector, representative vector**
    - **code words, code book**
- **Voronoi quantizer**
  - **nearest neighbor rule on the Euclidean metric**
  - **figure 9.12**
- **Learned Vector Quantization**
  - **a supervised learning technique**
  - **move Voronoi vector slightly in order to improve classification decision quality**

Learned Vector Quantization

- Suppose w_c is the closest to input x_i.
- Let C_w be the class of w_c
- Let C_{xi} be the class label of x_i

- If $C_w = C_{xi}$, then
  $w_c(n+1) = w_c(n) + a_n[x_i - w_c(n)]$
- otherwise
  $w_c(n+1) = w_c(n) - a_n[x_i - w_c(n)]$

- the other Voronoi vectors are not modified

Adaptive Pattern Classification

- Combination of Feature extraction and classification
- Feature extraction
  - unsupervised by SOFM
  - essential information contents of input data
  - data reduction
  - dimension reduction
- Classification
  - supervised scheme such as MLP

- Example
  - Figure 9.14

Hierarchical Vector Quantization

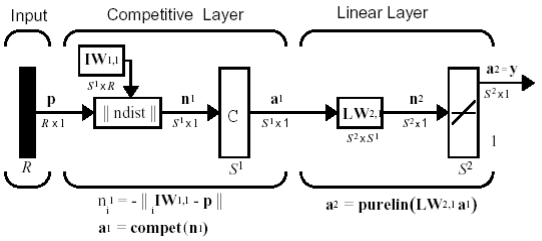
- Conventional VQ algorithm needs large amount of computation
  - input vector is compared with every code word
- multistage hierarchical vector quantization
  - tradeoff between accuracy and speed

MATLAB TOOLBOX

- net = newlvq(PR,S1,PC,LR,LF)
  - Description of function
    - LVQ networks are used to solve classification problems.
  - NET = NEWLVQ(PR,S1,PC,LR,LF) takes these inputs,
    - PR - Rx2 matrix of min and max values for R input elements.
    - S1 - Number of hidden neurons.
    - PC - S2 element vector of typical class percentages.
    - LR - Learning rate, default = 0.01.
    - LF - Learning function, default = 'learnlv1'.

    Returns a new LVQ network.
-

■ **The LVQ network architecture is shown below**



Where...

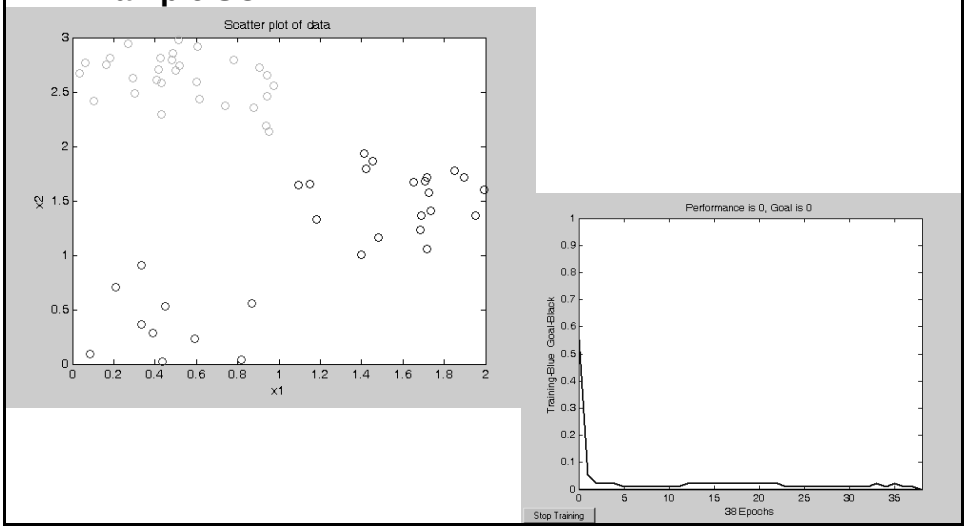
R = number of elements in input vector

S^1= number of competitive neurons

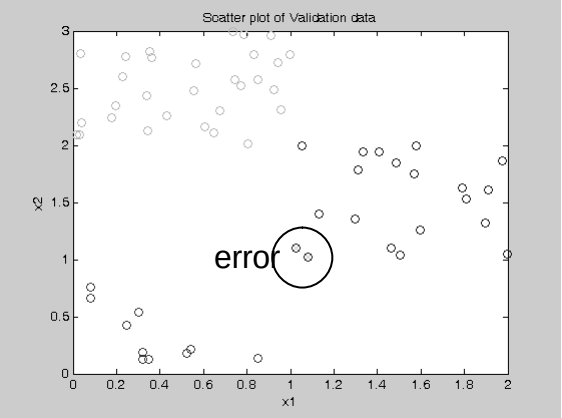
S^2= number of linear neurons

Example

■ **Example SOM :**



■ **Validation results error**



Lecture 10-69

■ **M_file**

E:\Matlab6.5\work\lvq_ex.m

File Edit View Text Debug Breakpoints Web Window Help

```
%classification with LVQ
%inputs are random varaible
P1=rand(20,2)';
P2=1+rand(40,2)';
P3=[0*ones(60,1) 2*ones(60,1)]'+rand(60,2)';
%split input to two sets:Learning and Validation
% in this example 50% for tranining and 50% for Validation
PL=[P1(:,1:10) P2(:,1:20) P3(:,1:30)];
PV=[P1(:,11:20) P2(:,21:40) P3(:,31:60)];
T=[ones(1,10) ones(1,20)*2 ones(1,30)*3];
plot(PL(1,1:10),PL(2,1:10),'or',PL(1,11:30),PL(2,11:30),'ob',PL(1,31:60),PL(2,31:60),'og')
xlabel('x1');ylabel('x2');title('Scatter plot of data')
net = newlvq(minmax(PL),6,[1/6 2/6 3/6]);
Tc = ind2vec(T);
net = train(net,PL,Tc);
%calculation of learning error
Y = sim(net,PL)
YL = vec2ind(Y)
eL=sum(abs(YL-T))
%calculation of validation error
Y = sim(net,PV)
YV = vec2ind(Y)eV=sum(abs(YV-T))
```

script Ln 23 Col 1