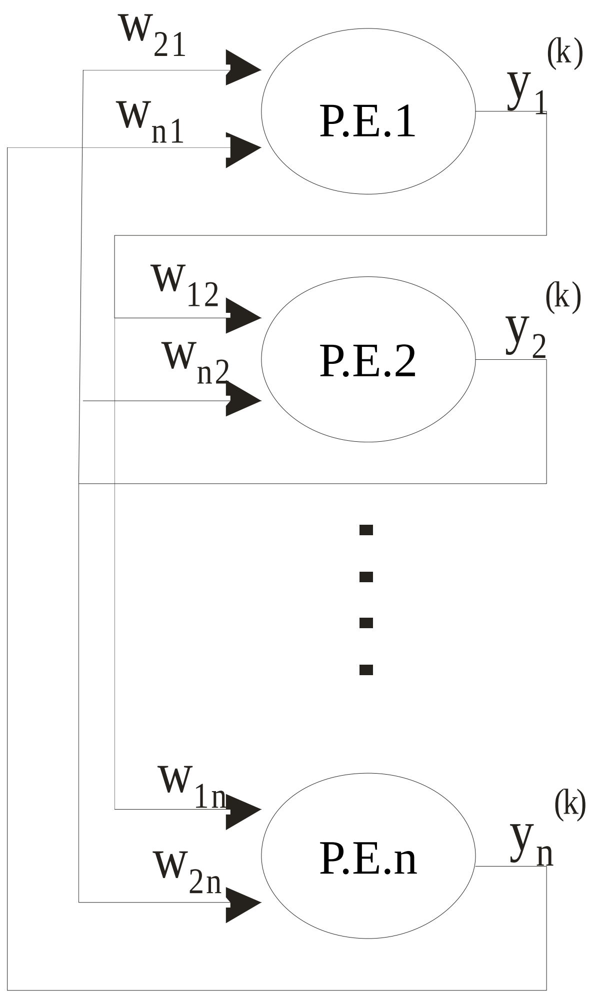
Hopfield Network– Recurrent Netorks



Hopfield Network with *n* Processing Elements

Auto-Associative Memory:
Given an initial *n* bit pattern
returns the closest stored (associated) pattern.
No P.E. self-feedback!

$$Dynamics: \; s_j^{(k)} = \sum_{i=1}^{n} w_{ij} y_i^{(k)}$$

$$y_j^{(k+1)} = f\left(s_j^{(k)}\right)$$

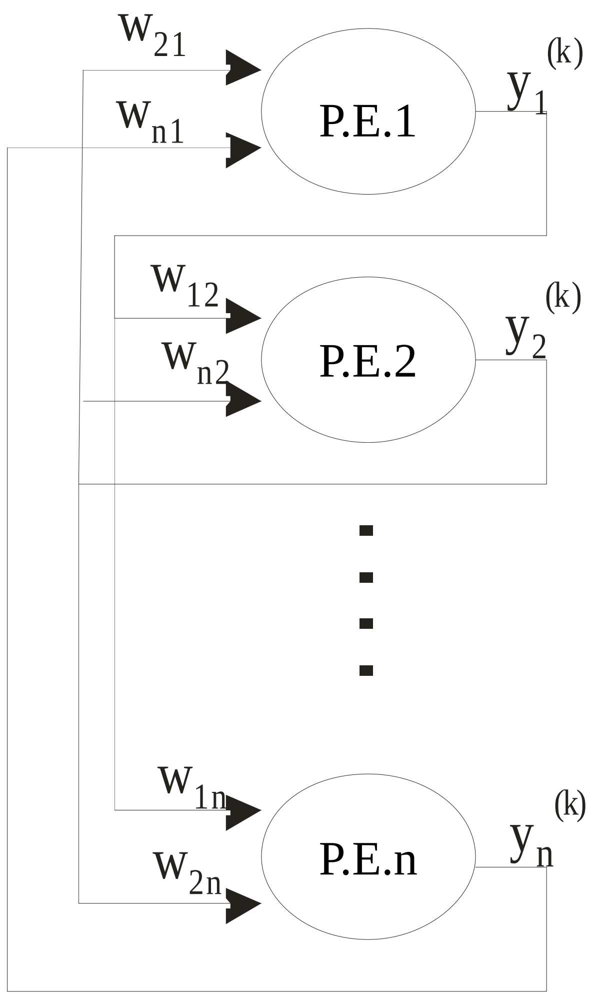
$$Network\ Initialization: \quad \mathbf{y}^{(0)} = \mathbf{x}$$

$$Output\ Vector: \quad \mathbf{y}^{(k)} = \left[y_i^{(k)}\right]$$

Binary activation function:

$$f(s_j) = \begin{cases} 1 \; if \;\; s_j > L_j \\ 0 \; if \;\; s_j < L_j \\ hold \;\; previous \; value, \; if \;\; s_j = L_j \end{cases}$$

Hopfield Network...

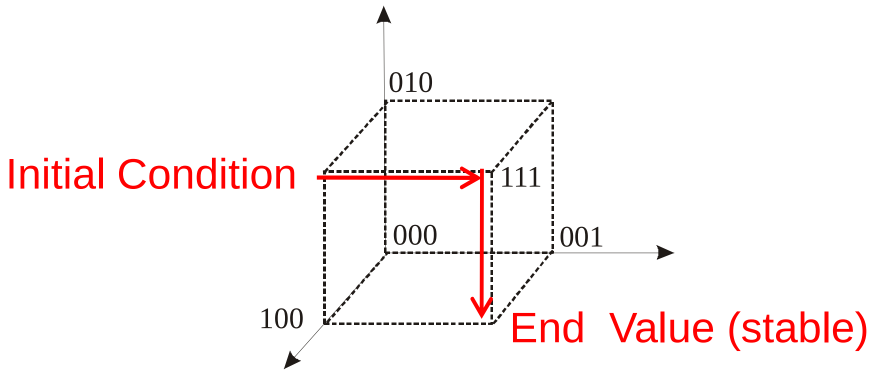


Hopfield Network with *n* Processing Elements

- **Fast training and fast data recovery**
- IIR system with no input (only I.C.)
- Guaranteed stability
- Good for VLSI implementation

-Operating Forms (firing order)

- Asynchronous
- Synchronous
- Sequential



Possible Hopfield Network states (8) with 3 Processing Elements
(Illustration of a typical recovery state evolution. From I.C. to E.V.)

Hopfield Network...

Learning:

The patterns to be stored in the associative memory are chosen a priori.

m distinct patterns. Each of the form:

$$A_p = \lfloor a_1^p \quad a_2^p \quad \dots \quad a_n^p \rfloor, \quad \text{with } a_i^p = 0 \text{ or } 1. \ (L = 0, \textit{usually})$$

$$w_{ij} = \sum_{p=1}^{m} (2a_i^p - 1)(2a_j^p - 1)$$

Obs: $(2a_i^p - 1)$ converts 0/1 to –1/+1

w_{ij} is incremented by 1 if $a_i^p = a_j^p$ otherwise it is decremented
Procedure is repeated for each *i,j* for every A_p.

Learning is analogous to *reinforcement learning*

Hopfield Network - Example

Patterns to be stored as 3x3 matrices:

a_1	a_2	a_3
a_4	a_5	a_6
a_7	a_8	a_9

1		
1		
1	1	1

1	1	1
	1	
	1	

	1	
1	1	1
	1	

Symbol	Training Vector
L	$A_1 = [1\ 0\ 0\ \ 1\ 0\ 0\ \ 1\ 1\ 1]$
T	$A_2 = [1\ 1\ 1\ \ 0\ 1\ 0\ \ 0\ 1\ 0]$
+	$A_3 = [0\ 1\ 0\ \ 1\ 1\ 1\ \ 0\ 1\ 0]$

$$w_{ij} = \sum_{p=1}^{m} (2a_i^p - 1)(2a_j^p - 1)$$

Weigth Matrix

$$W = \begin{bmatrix} 0 & -1 & 1 & -1 & -1 & -3 & 1 & 1 & 1 \\ -1 & 0 & 1 & -1 & 3 & 1 & -3 & 1 & -3 \\ 1 & 1 & 0 & -3 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -3 & 0 & -1 & 1 & 1 & 1 & 1 \\ -1 & 3 & 1 & -1 & 0 & 1 & -3 & 1 & -3 \\ -3 & 1 & -1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & -3 & -1 & 1 & -3 & -1 & 0 & -1 & 3 \\ 1 & 1 & -1 & 1 & 1 & -1 & -1 & 0 & -1 \\ 1 & -3 & -1 & 1 & -3 & -1 & 3 & -1 & 0 \end{bmatrix}$$

Hopfield Network - Example

New pattern presented to the trained network:

1		1
1		
	1	1

$\mathbf{x} = \mathbf{y}^{(0)} = [\ 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1\ ]$

$$W = \begin{bmatrix} 0 & -1 & 1 & -1 & -1 & -3 & 1 & 1 & 1 \\ -1 & 0 & 1 & -1 & 3 & 1 & -3 & 1 & -3 \\ 1 & 1 & 0 & -3 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -3 & 0 & -1 & 1 & 1 & 1 & 1 \\ -1 & 3 & 1 & -1 & 0 & 1 & -3 & 1 & -3 \\ -3 & 1 & -1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & -3 & -1 & 1 & -3 & -1 & 0 & -1 & 3 \\ 1 & 1 & -1 & 1 & 1 & -1 & -1 & 0 & -1 \\ 1 & -3 & -1 & 1 & -3 & -1 & 3 & -1 & 0 \end{bmatrix}$$

Sequential operation of the network:

Fired P.E.	P.E. Sum	P.E. Output	New output vector
1	2	1	1 0 1 1 0 0 0 1 1
2	-3	0	1 0 1 1 0 0 0 1 1
3	-4	0	1 0 **0** 1 0 0 0 1 1
4	1	1	1 0 0 1 0 0 0 1 1
5	-4	0	1 0 0 1 0 0 0 1 1
6	-4	0	1 0 0 1 0 0 0 1 1
7	4	1	1 0 0 1 0 0 **1** 1 1
8	0	1	1 0 0 1 0 0 1 1 1
9	4	1	1 0 0 1 0 0 1 1 1
1	2	1	1 0 0 1 0 0 1 1 1
2	-8	0	1 0 0 1 0 0 1 1 1

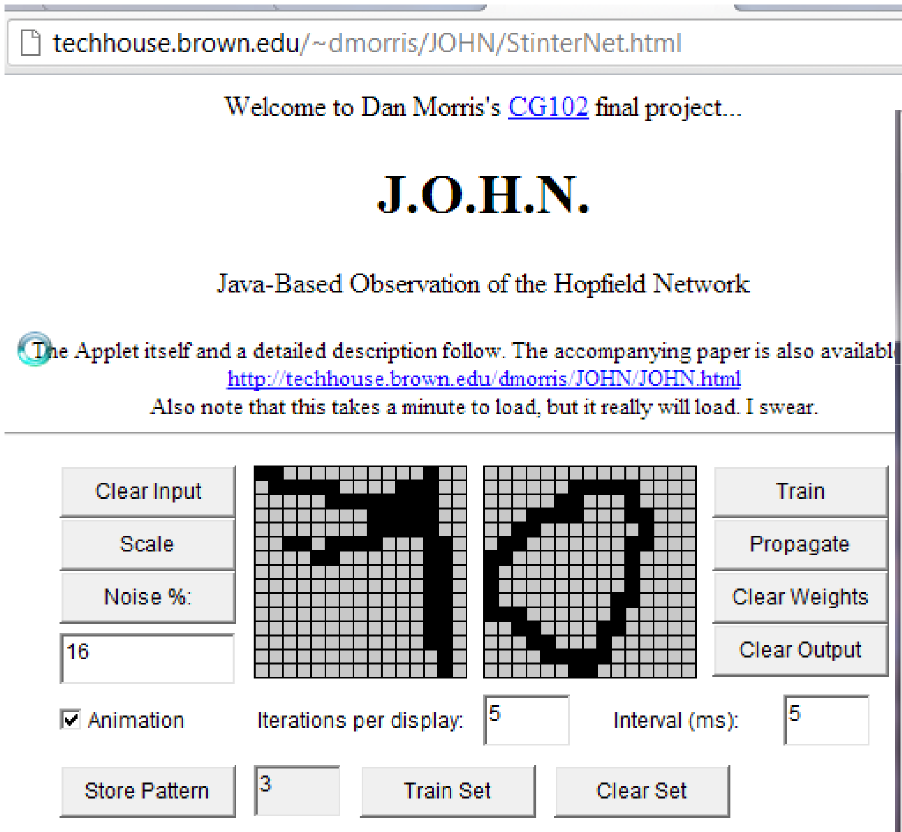
Convergence to “L” Pattern

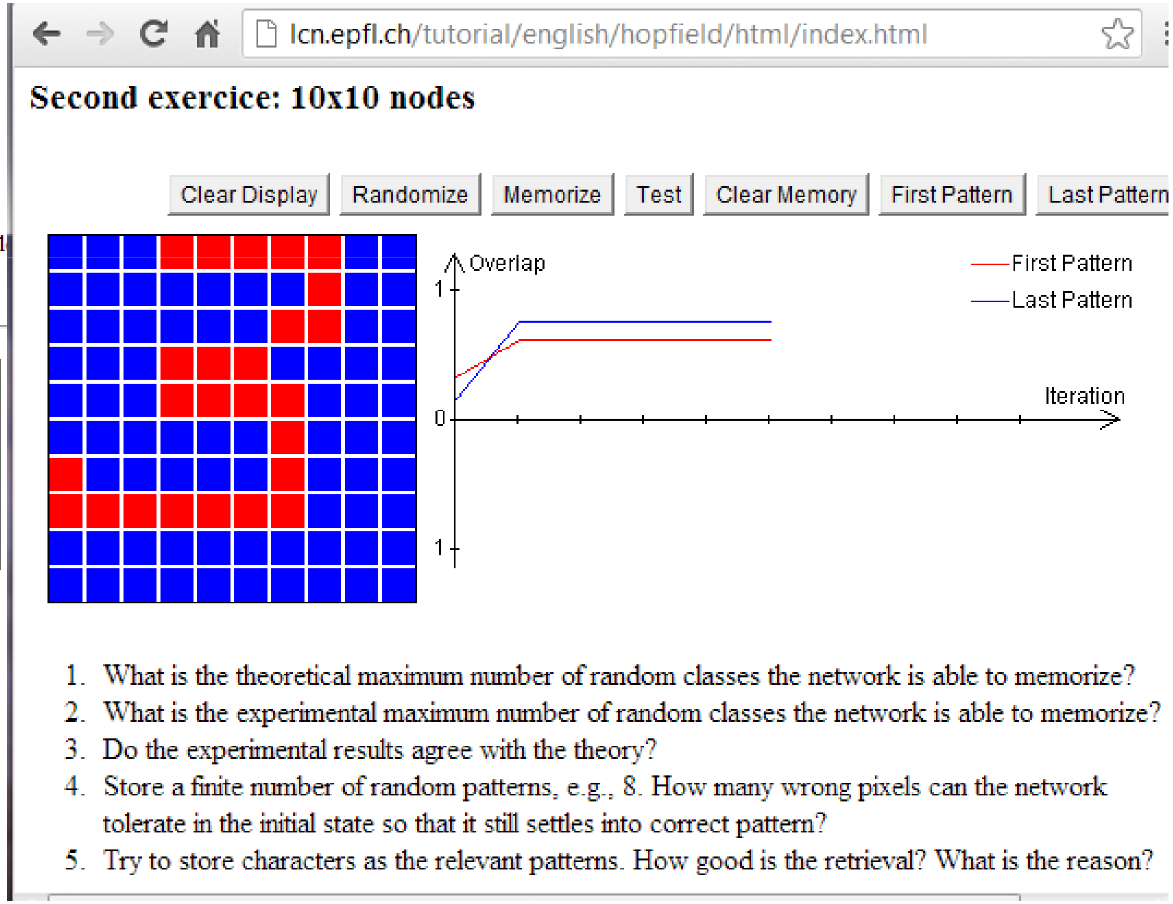
Remember – Binary activation function, $L_j = 0$:

$$f(s_j) = \begin{cases} 1 \text{ if } s_j > 0 \\ 0 \text{ if } s_j < 0 \\ \text{hold previous value, if } s_j = 0 \end{cases}$$

Hopfield Network – java demos

Demonstrations available in the www, e.g.:





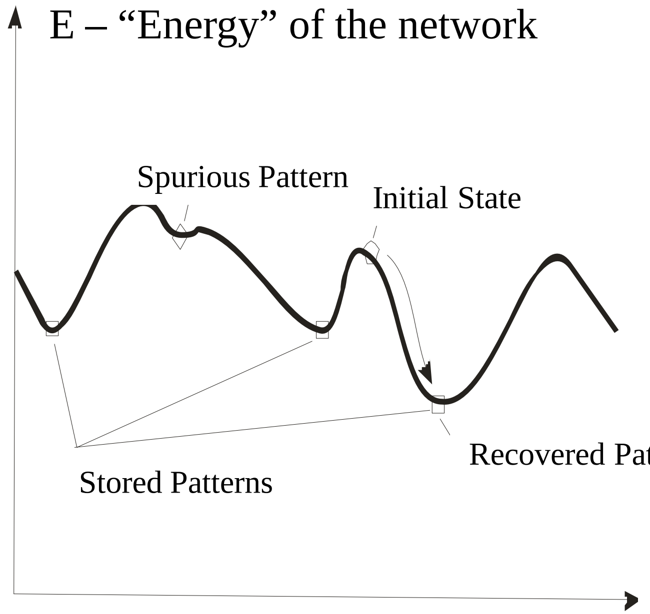
Hopfield N. – final considerations

Stability proof – Cohen and Grossberg, 1983.

W symmetric with zero diagonal

"Energy funcition" always decreases.

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} y_i y_j - \sum_j y_j L_j$$



Typical Energy and Patterns illustration for Hopfield Newtworks

Hopfield Network Limitations:

- Not necessarily the closest pattern is returned.
- Differences between patterns. Not all patterns have equal emphasis (size of attraction basins).
- Spurious patterns, i.e., patterns evoked that are not part of the stored set.
- Maximum number of stored patterns is limited.

$m \leq 0{,}5n/\log n$, m patterns, n bits network

Radial Basis Functions

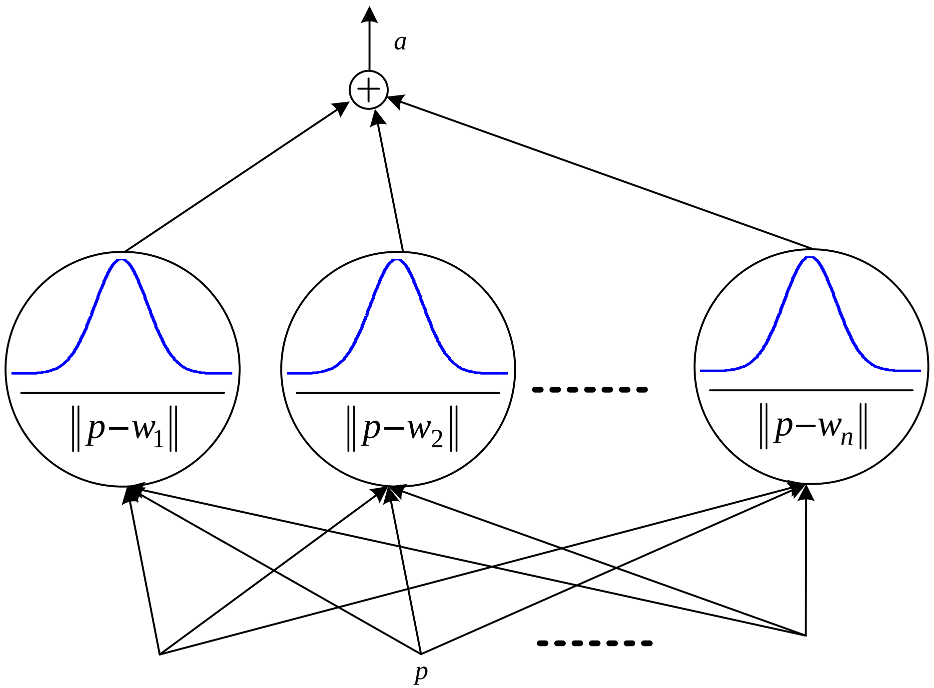
- Moody & Darken, 1989,...
- Function Approximators
- Inspiration: sensoricc overlapped reception fields in the cortex
- ***Localized activity*** of the processing elements

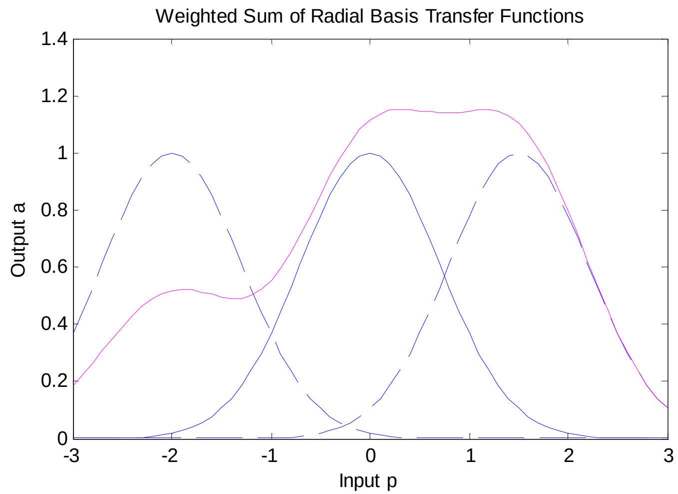
$$a_i = e^{-\frac{\|x_i - \mu_i\|}{\sigma_i^2}}$$

Gaussian
(Average, Variance)

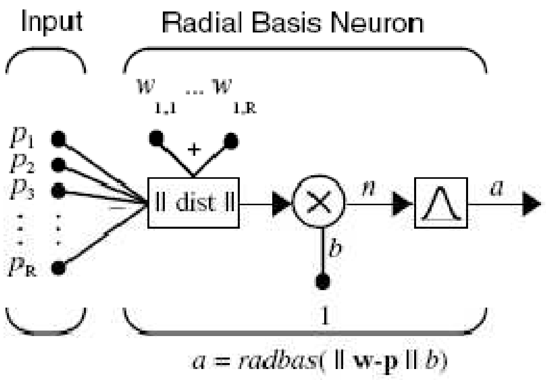
$$a_i = e^{-\|p_i - w_i\| b}$$

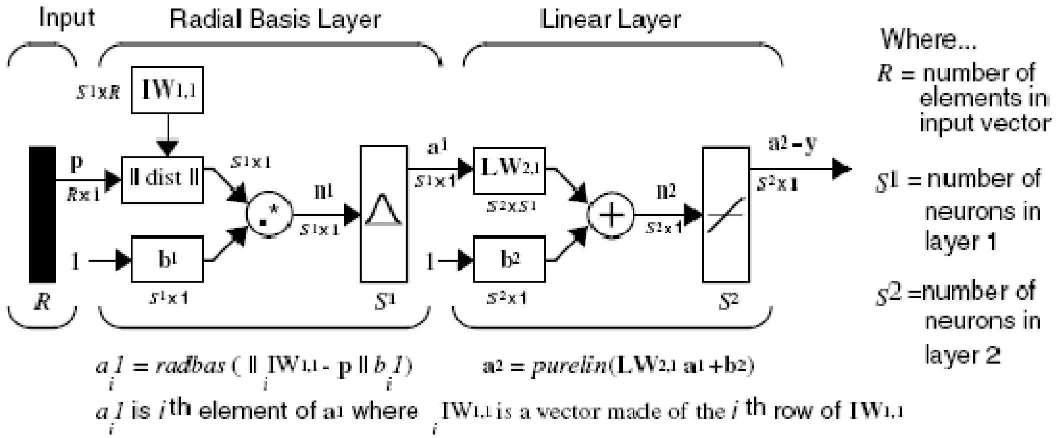
ml:
w – weigth
b – bias

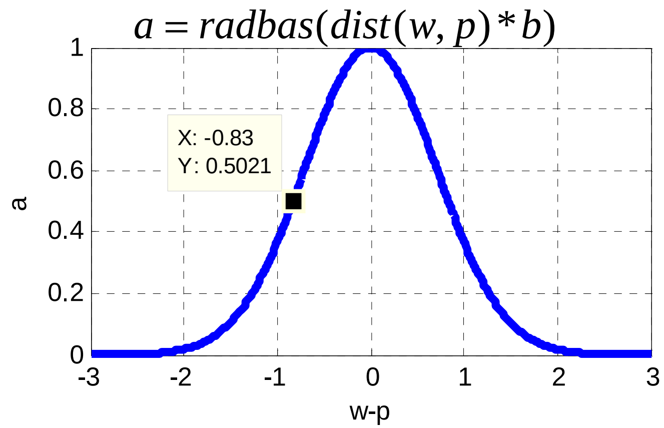




Radial Basis Functions...



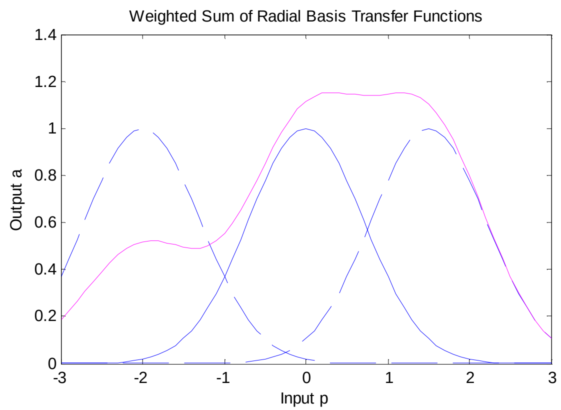




MatLab Implementation

[net,tr] = newrb(P,T,GOAL,SPREAD,MN)

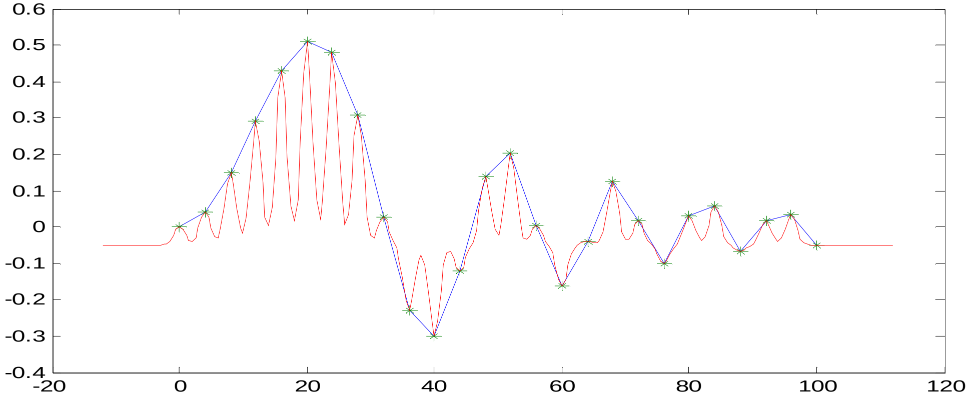
P - RxQ matrix, Q input vectors (“pattern”),
T - SxQ matrix, Q objective vectors (“target”),
GOAL - desired mean square error, default = 0.0,
SPREAD - radial basis function spread, default = 1.0,
MN - Maximum number of neurons, default is Q.



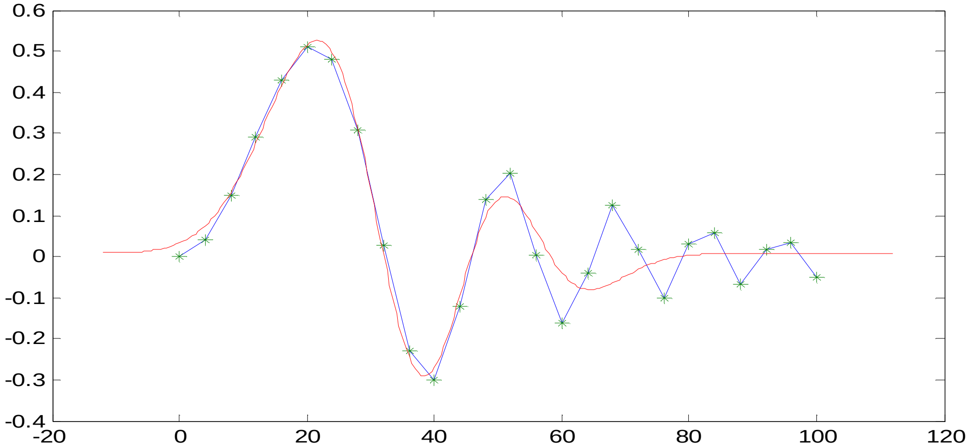
Radial Basis Functions...

Learning: add neurons incrementally

- Where? To obtain the largest quadratic errror reductions at each step $\rightarrow \min \sum e^2 = 0$

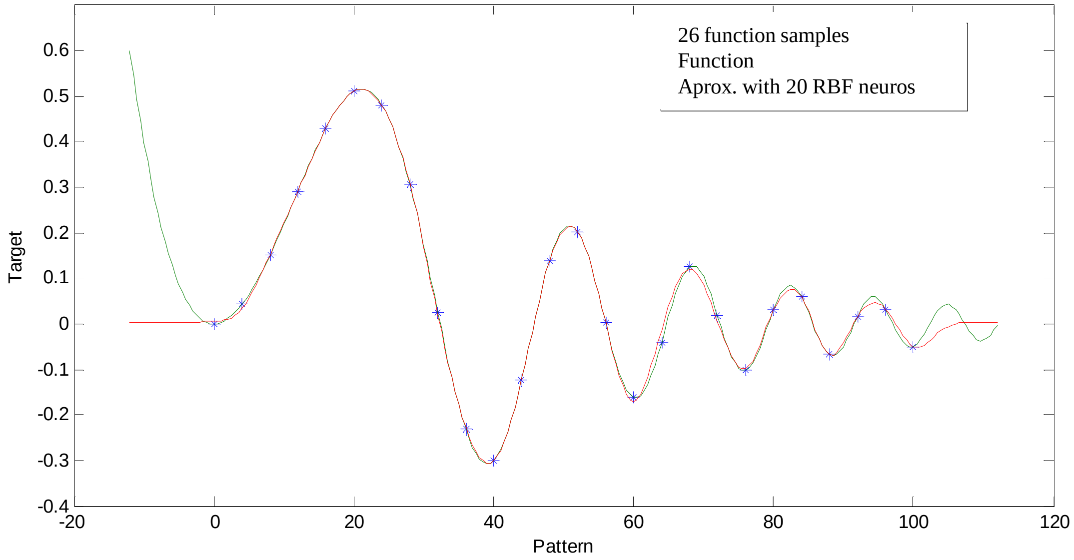


spread → to low
- Good **fitting** (at the training points)!
- Bad **interpolation!**



spread → to high
- Good **fitting at low frequencies**
- Good **interpolation in some ranges!**

Radial Basis Functions...



Heuristics:

$$|x_{i+1} - x_i| < SPREAD < |x_{\max} - x_{\min}|$$

spread → OK
- Good **fitting**!
- Good **interpolation!**

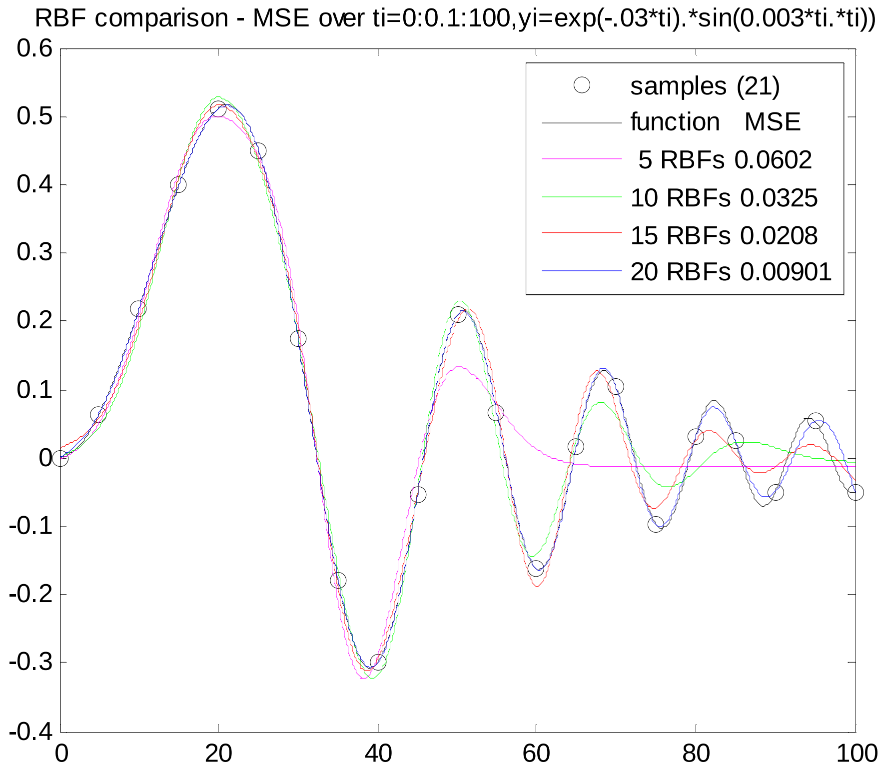
-Bad extrapolation
(is a very difficult task)

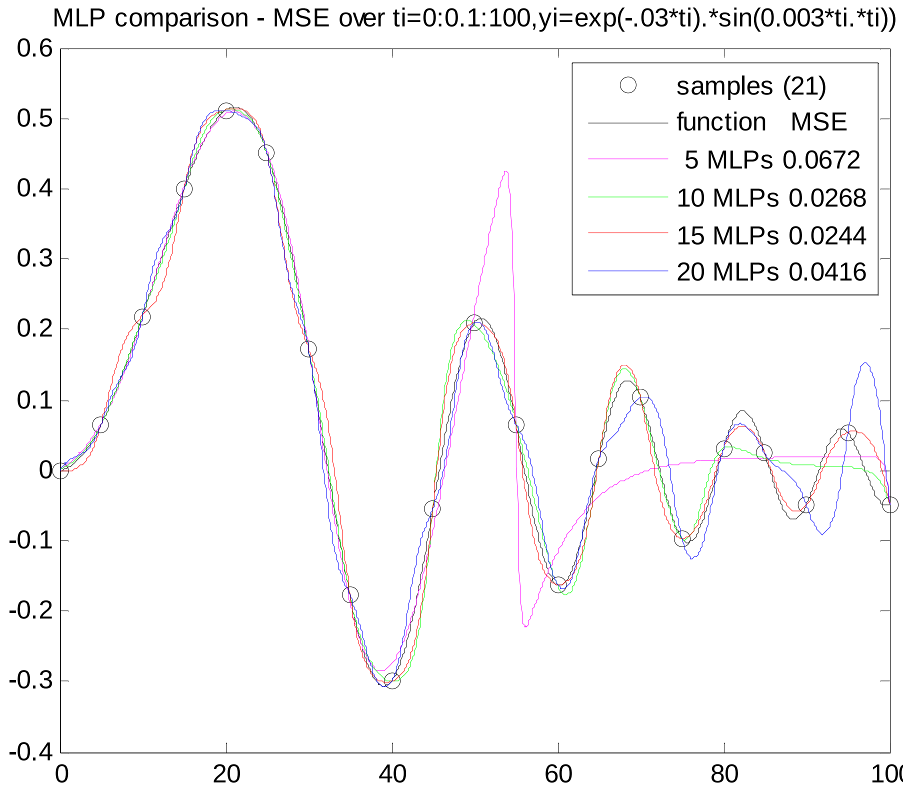
Conclusions

- Faster training faster, but uses more neurons than MLP.
- Incremental Training, new points can be learned without losing prior knowledge.
-You can use *a priori* knowledge to locate neurons (which is not possible in a MLP).

- Fixed spread – Incremental traning → **suboptimal solution**!!

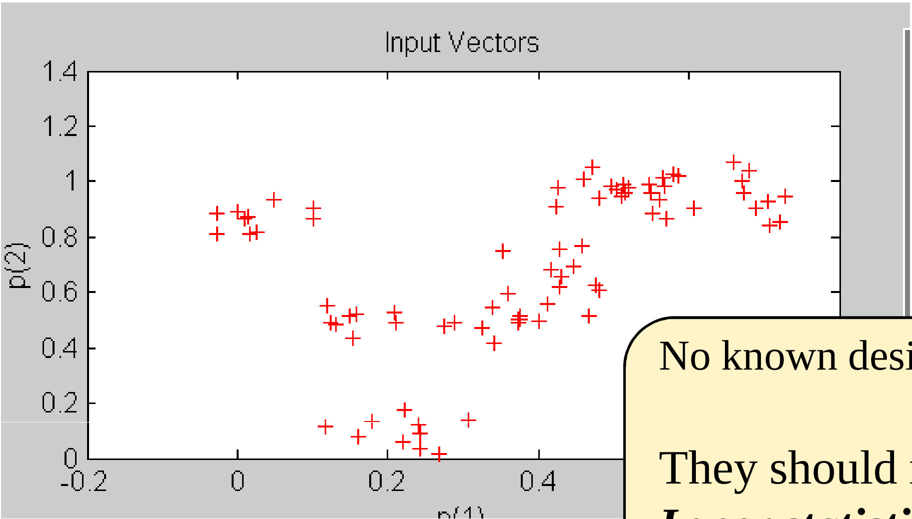
Comparison RBF x MLP





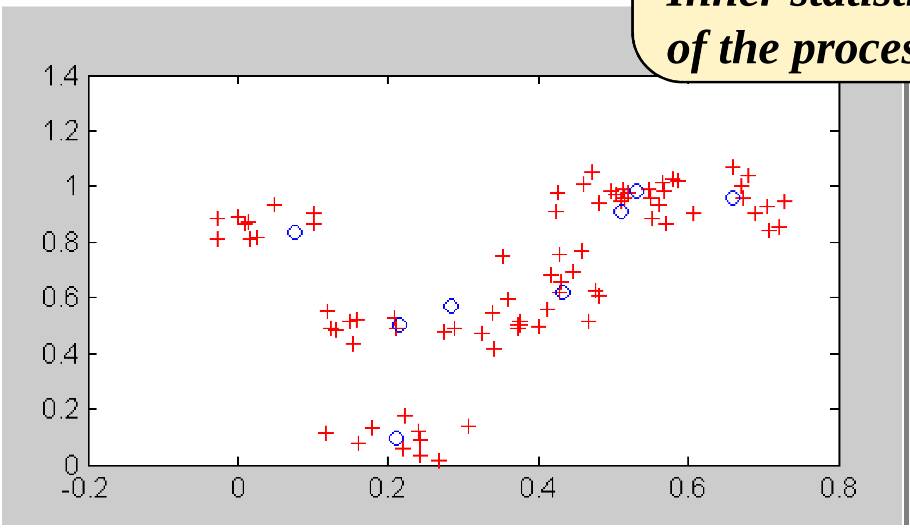
RBF – more neurons better fitting → best solution newrbe (exact fitting!)
MLP – too much neurons → worse fitting (bad interpolation)

Unsupervised Learning



No known desired Code Vectors

They should reflect the ***Inner statistical distribution of the process***



Competitive Layer

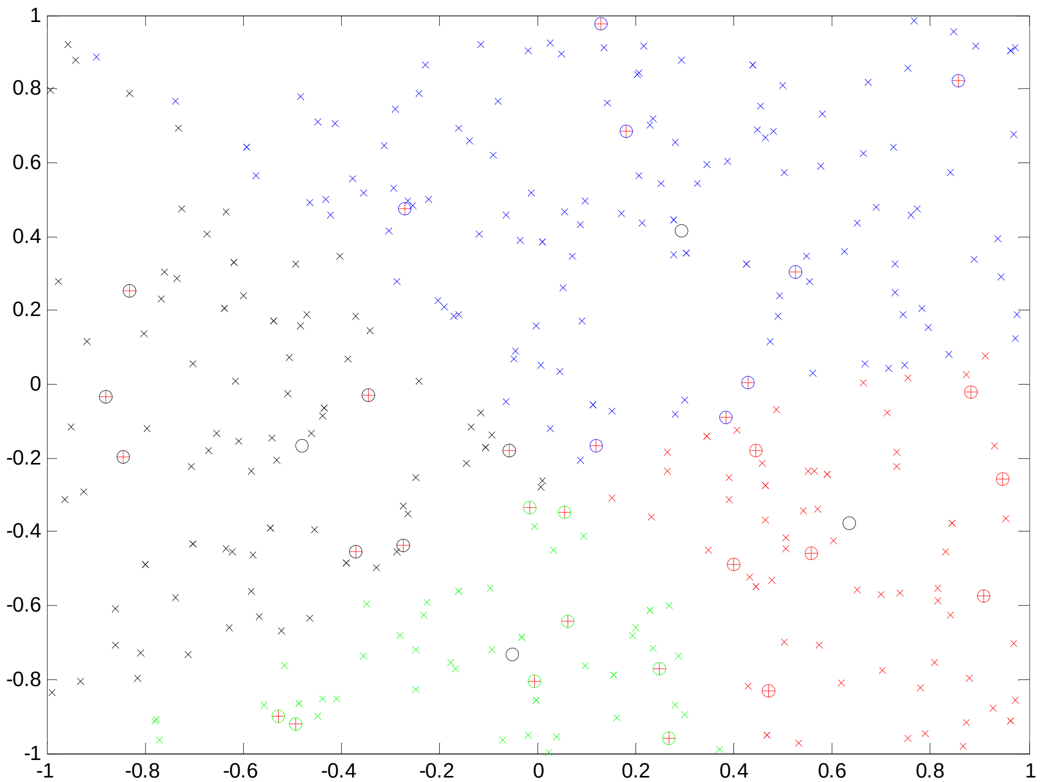
Find vector codes that describe the data distribution

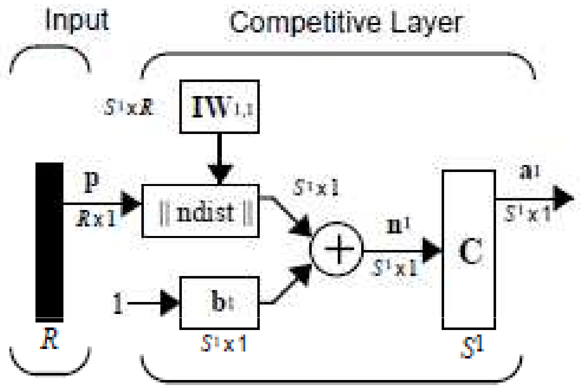
Used in Data Compression

Example:
Code symbols that will be transmitted over a communication channel.

For the comprehension the variability of the signal is considered as "noise", and so, discarded.

Competitive Layer



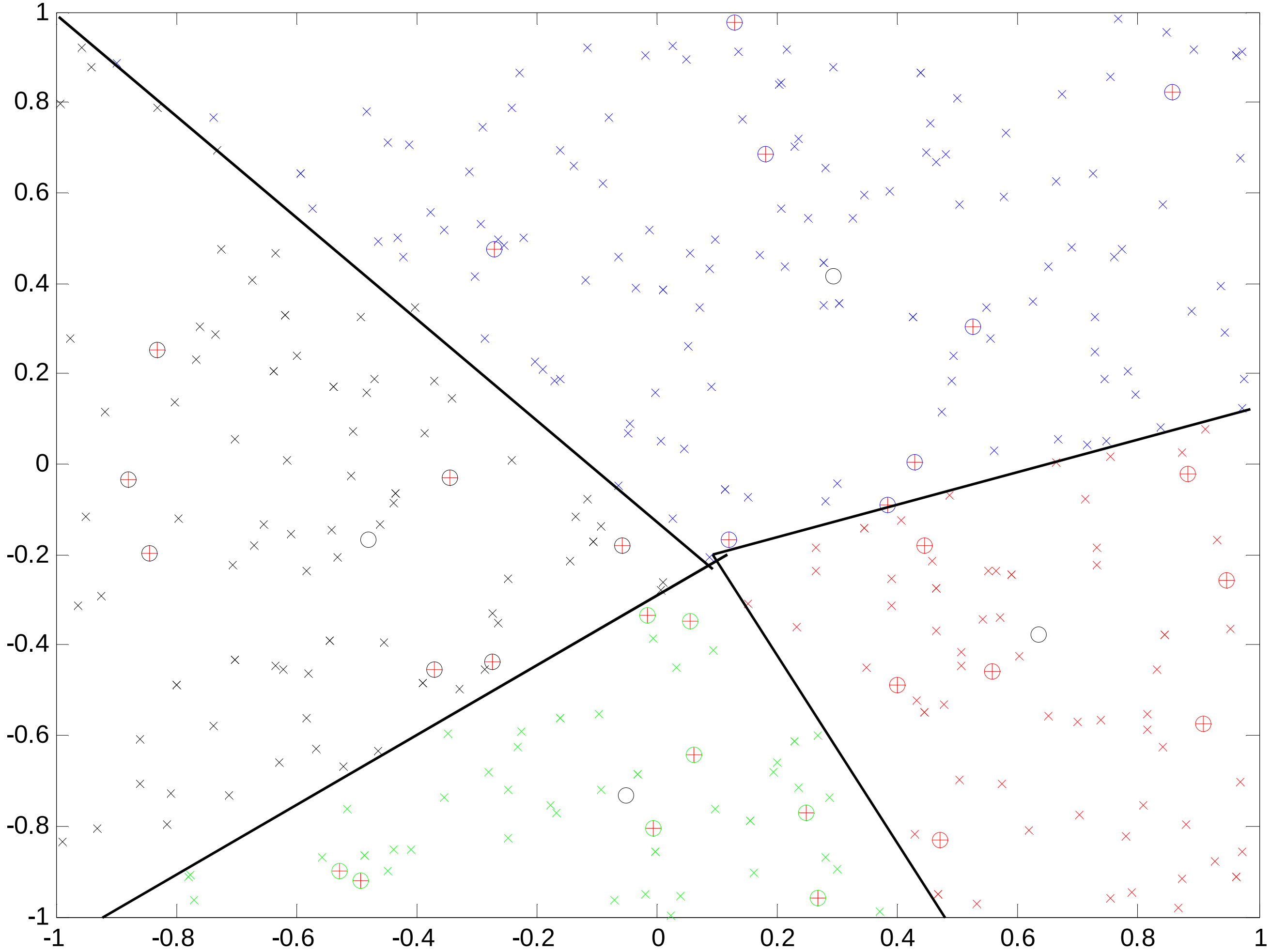


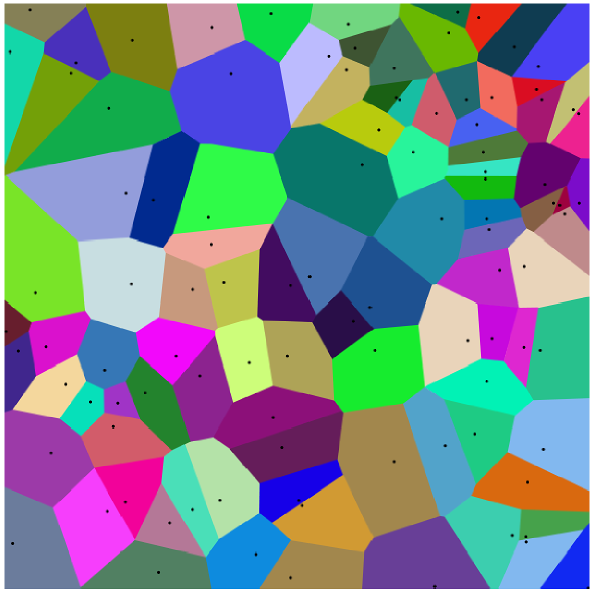
o code vectors

⊕ training vectors

x test vectors

Borders





(Voronoi Diagram)

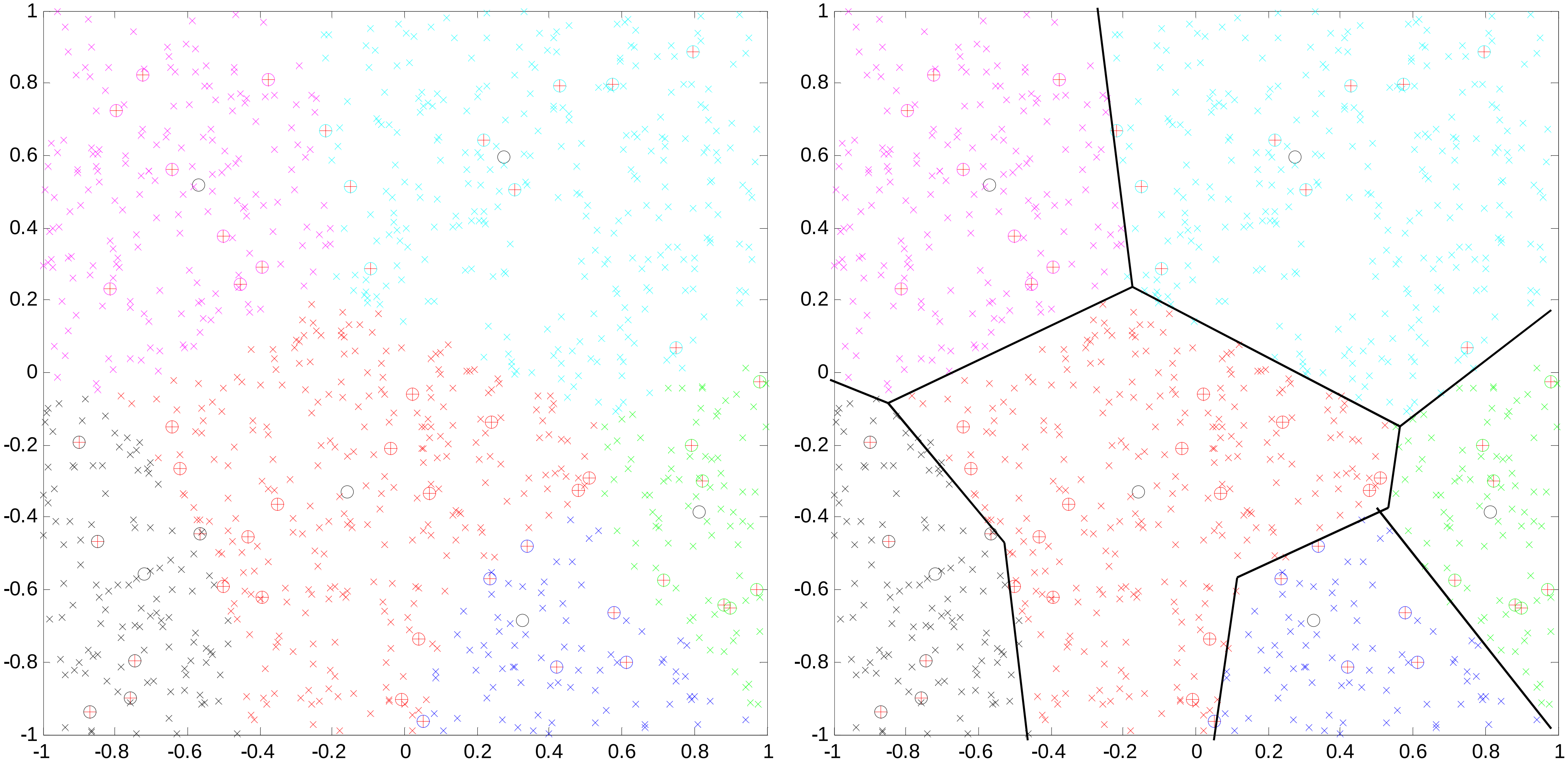
o code vectors

⊕ training vectors

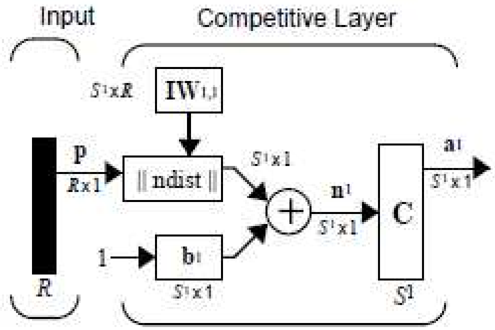
x test vectors

Ex. Classification Borders

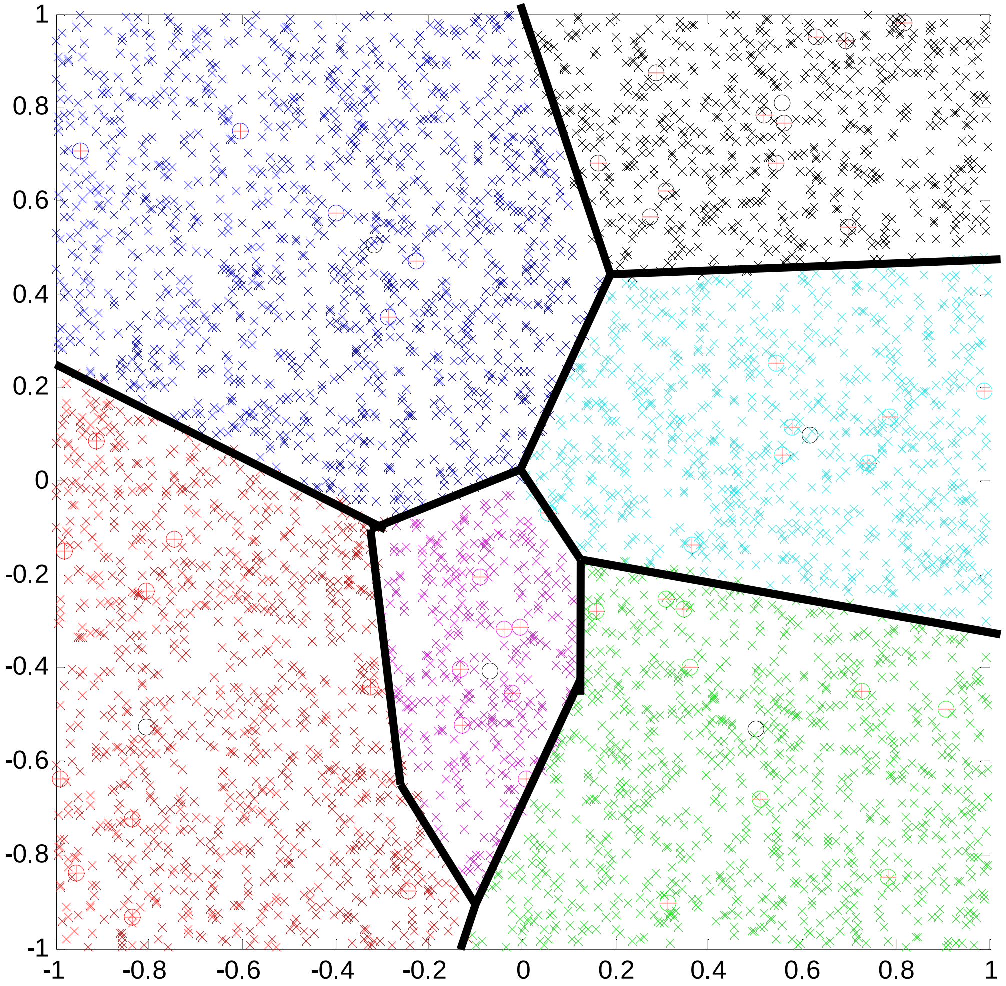
o code vectors
⊕ training vectors
x test vectors



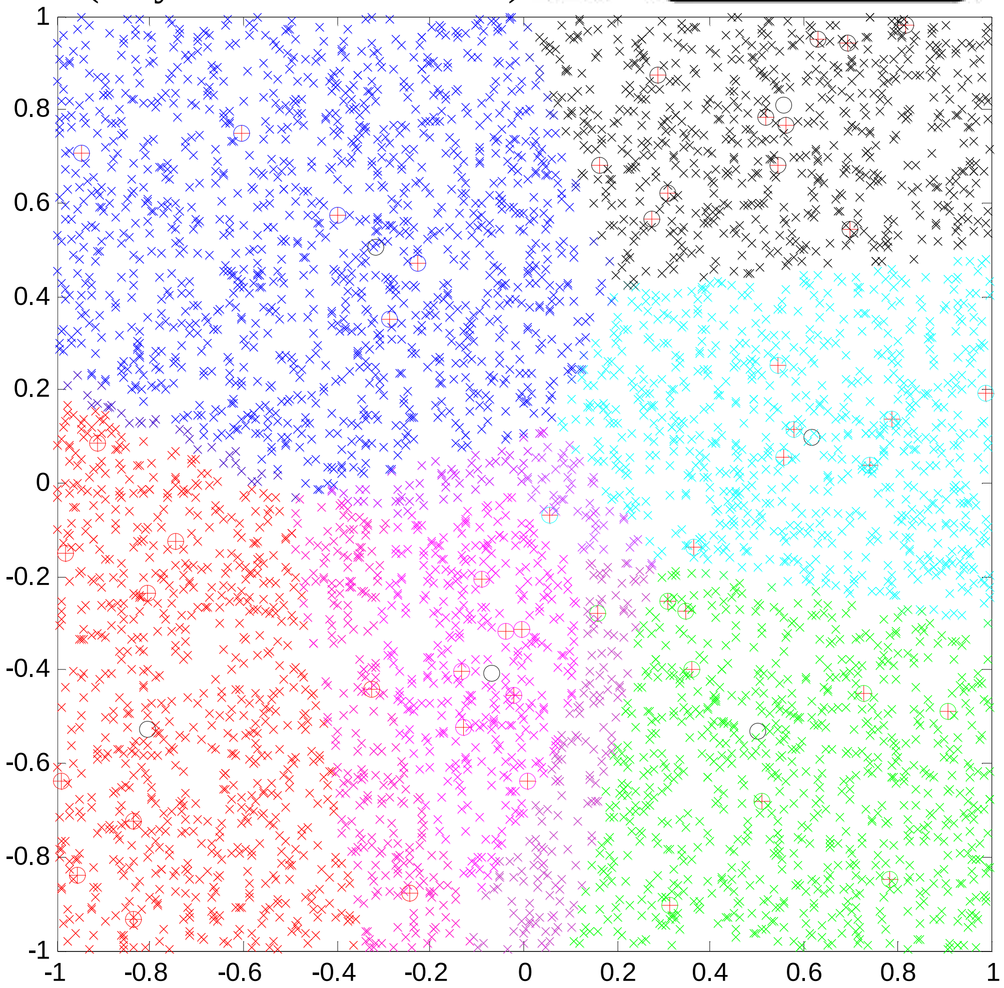
Bias



Bias adjustment to to help “weak” neurons



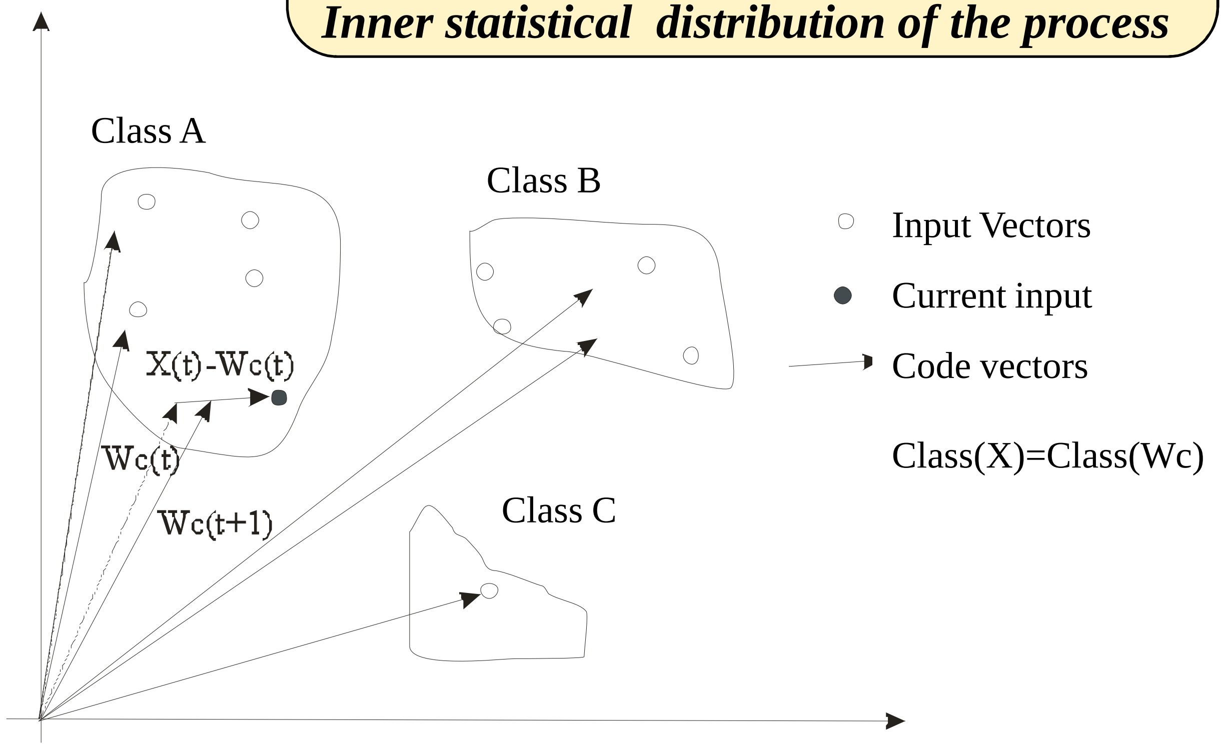
Bias = 0
(only Euclidean distance)



Learning Vector Quantization

No known desired Code Vectors

Data points belong to Classes
Code Vectors should reflect the
Inner statistical distribution of the process



LVQ1, LVQ2.1, LVQ3, OLVQ

"Enhanced Algorithms"

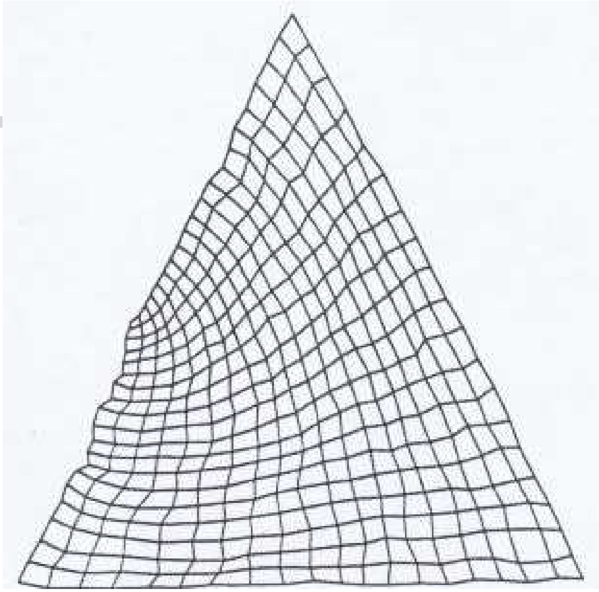
-Dead neurons
-Neighborhood definition

Self Organizing Maps

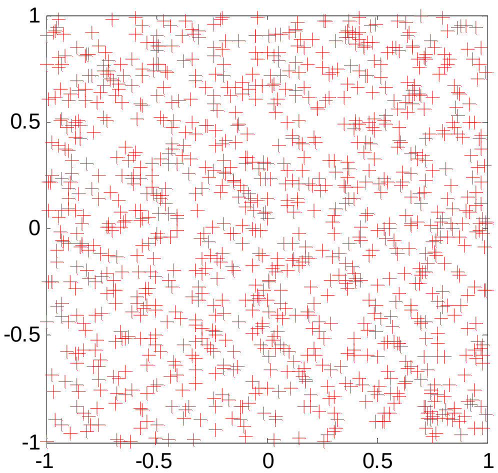
Kohonen, 1982 – Unsupervised learning

One active layer with *neighborhood* constrains

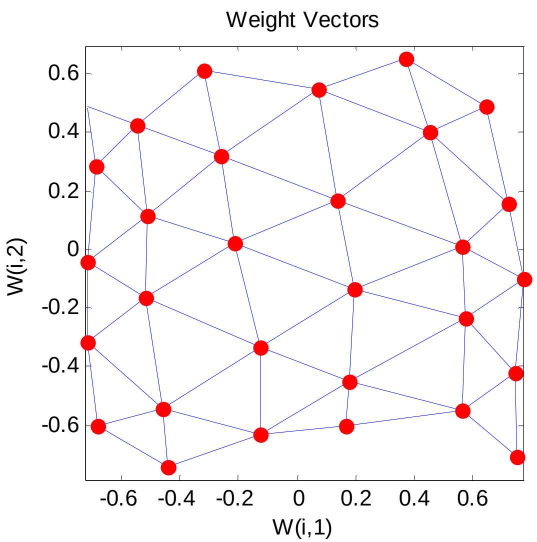
Code Vectors should reflect the
Inner statistical distribution of the process

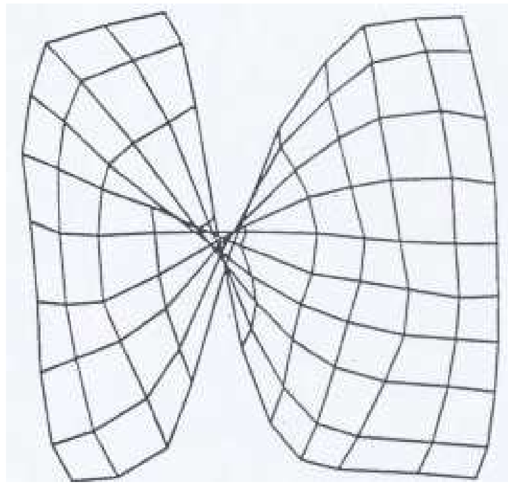


Triangular distribution



SOM Map





Weird unsuccessful training

ANN General Characteristics

- Positive
  - Learning
  - Parallelism
  - Distributed knowledge
  - Fault Tolerant
  - Associative Memory
  - Robust against Noise
  - No exhaustive modelling
- Negative
  - Knowledge acquisition only by learning
    (“E.g., Wich topology is best suit?)
  - Introspection is not possible
    (“What is the contribution of this neuron?)
  - The logical inference is hard to obtain
    (“Why this output for this situation?”)
  - Learning is slow
  - Very sensitive to initial conditions

To obtain successful ANN
a good process knowledge is
recommended in order to design
experiments that produce useful data sets!

There is no free lunch!