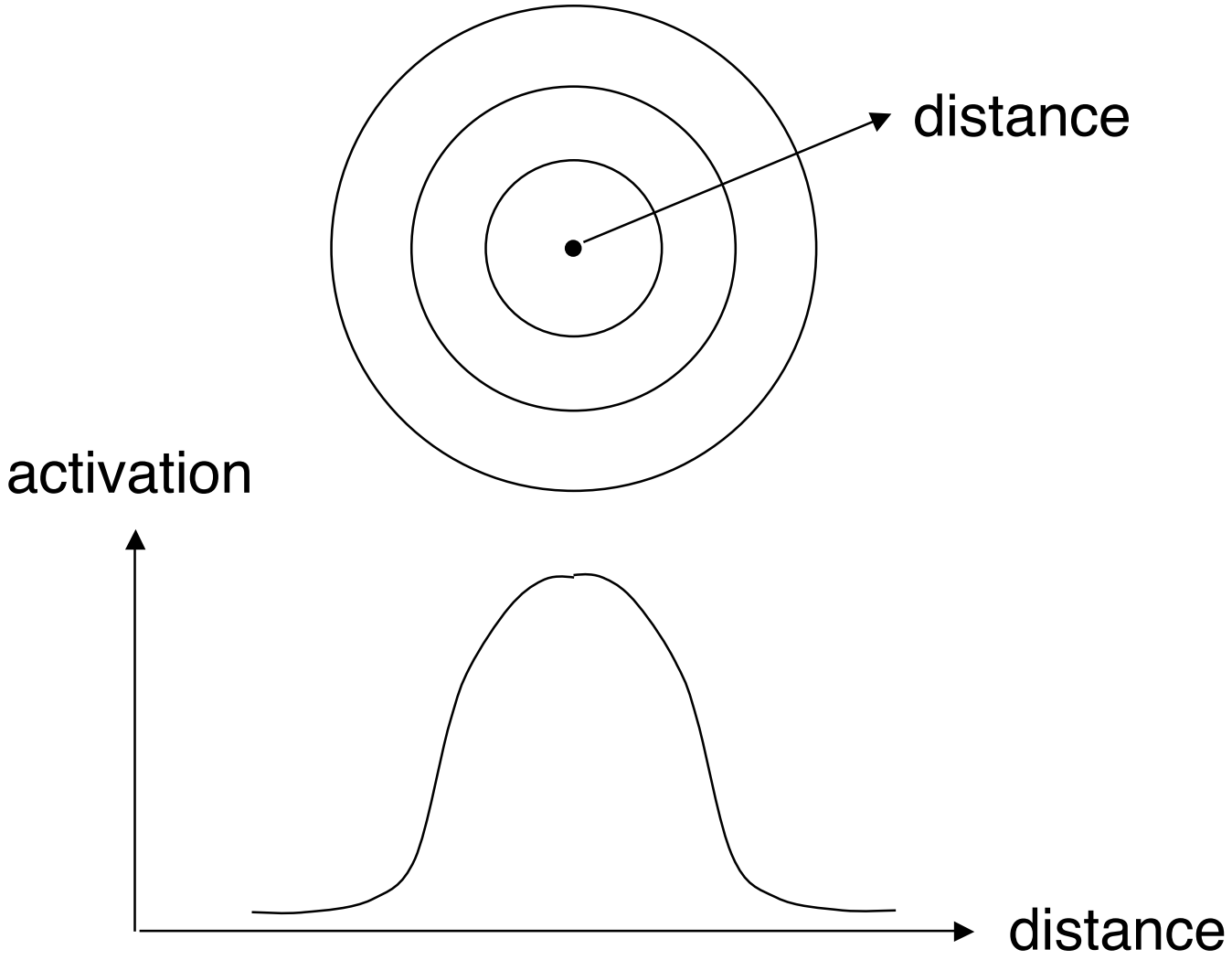
Radial Basis Function Networks

Radial Basis Functions

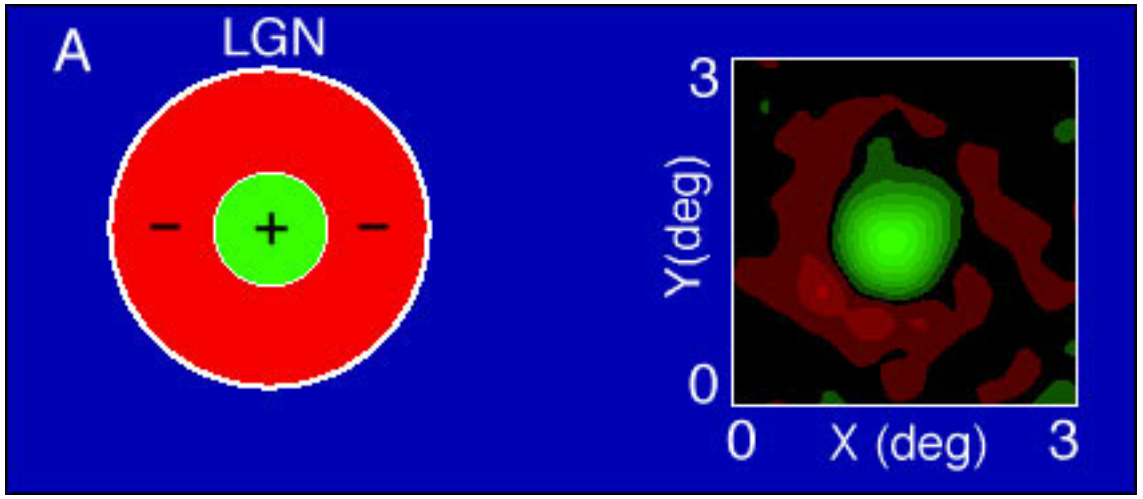
- In contrast to sigmoidal functions, radial basis functions have **radial symmetry** about a *center* in n-space (n = # of inputs).
- The **farther** from the center the input is, the **less** the activation.
- This models the "**on-center off-surround**" phenomenon found in certain **real neurons** in the visual system, for example.

On-Center, Off-Surround



On-Center response captured in a lab experiment

(from http://ferguson.bvu.edu/Perception/Visual_system.html)



(LGN = lateral geniculate nucleus, see next slide)

LGN description,

from http://www.science.gmu.edu/~nbanerje/csi801/report_html.htm

LGN is a folded sheet of neurons (1.5 million cells), about the size of a credit card but about three times as thick, found on each side of the brain.

The ganglion cells of the LGN transform the signals into a temporal series of discrete electrical impulses called action potentials or spikes.

The ganglion cell responses are measured by recording the temporal pattern of action potentials caused by light stimulation.

(continued)

LGN description,

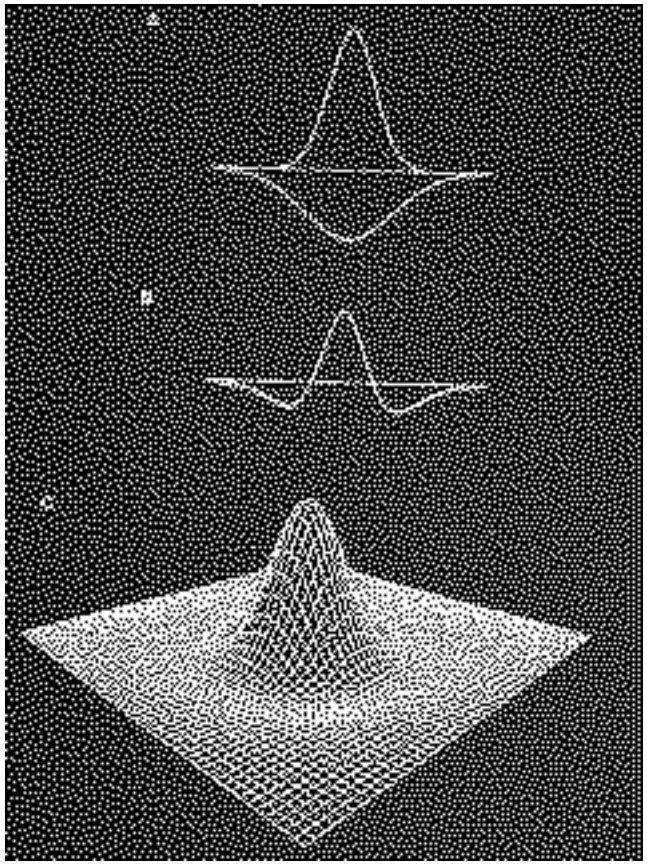
from http://www.science.gmu.edu/~nbanerje/csi801/report_html.htm

The receptive fields of the LGN neurons are **circularly symmetric** and have the same **center-surround** organization.

The algebraic sum of the center and surround mechanisms has a vague resemblance to a **sombrero** with a tall peak, so this model of the receptive field is sometimes called "**Mexican-hat model**."

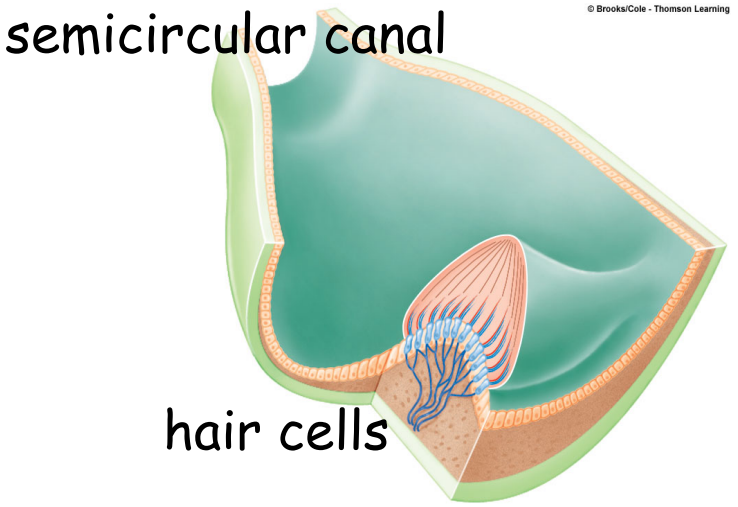
When the spatial profiles of center and surround mechanisms can be described by **Gaussian** functions the model is referred to as the "**difference-of-Gaussians**" model.

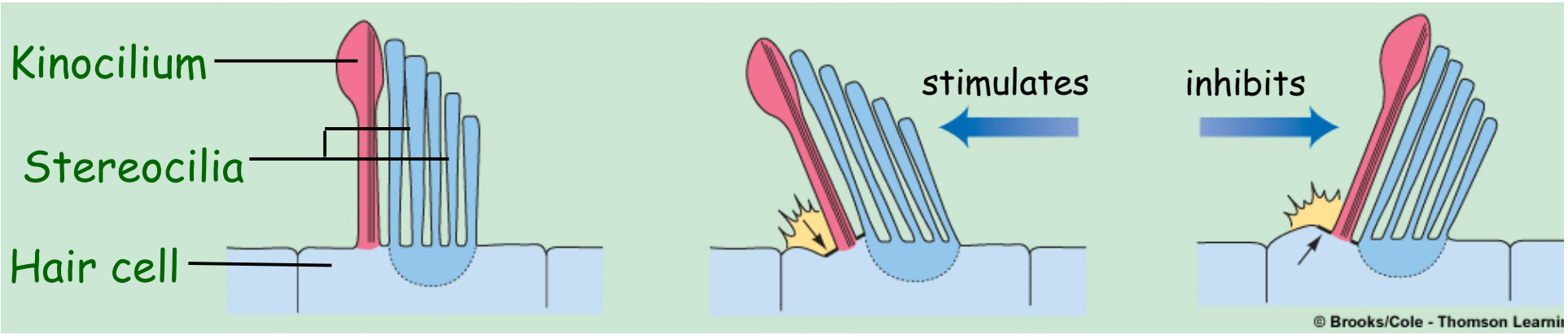
3-D depiction of 2-D on-center response ("sombrero hat")



On-Center Behavior

Also occurs in the ear: sensitivity to tones in the **cochlear stereocilia cells**.





Possible Applications

- Face recognition
- Odor sensing
- Color image classification
- Time series applications, forecasting

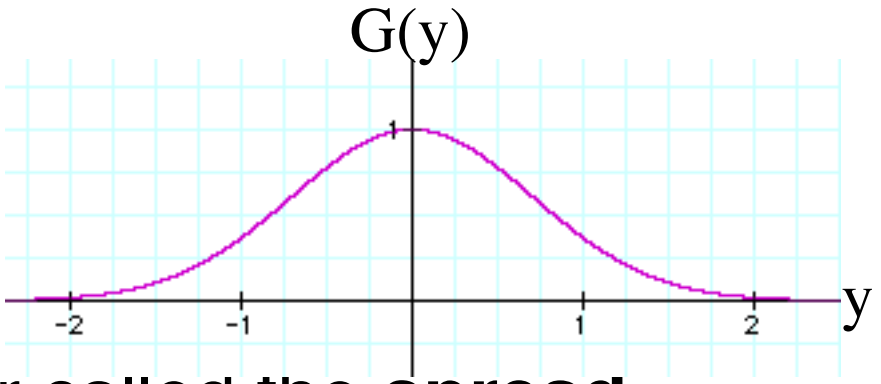
Modeling

- $\varphi_i(\mathbf{x}) = G(\ \| \mathbf{x} - \mathbf{c}_i \| \ )$

  where G decreases away from 0 and $\mathbf{c}_i$ is the **center**.

- Example: Gaussian:

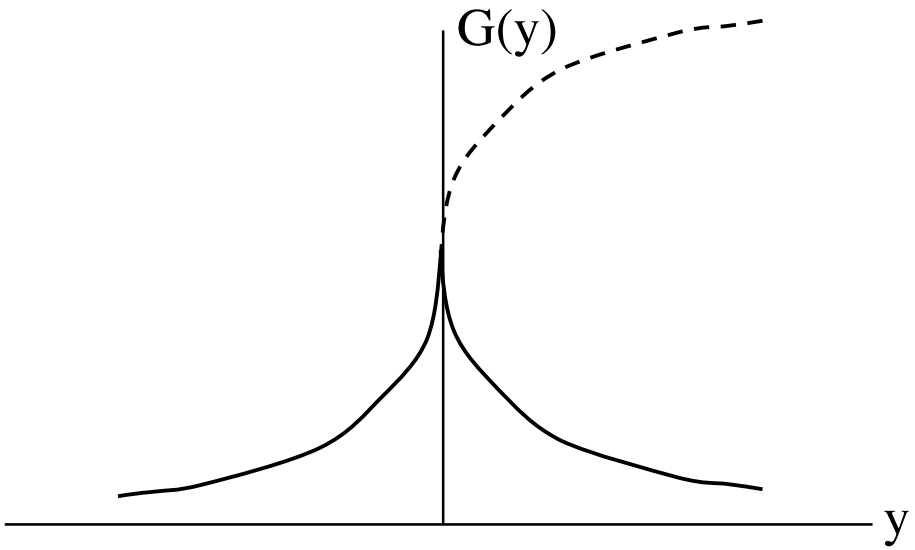
  $G(y) = \exp(-y^2/\sigma^2)$



  where σ is a parameter called the **spread**, which indicates the **selectivity** of the neuron.

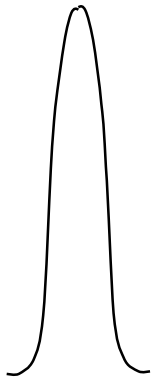
Other RBF Examples

- G(y) = 1/sqrt($y^2 + \sigma^2$)
- G(y) = 1/(1+exp(ay^2)) “reflected sigmoid”

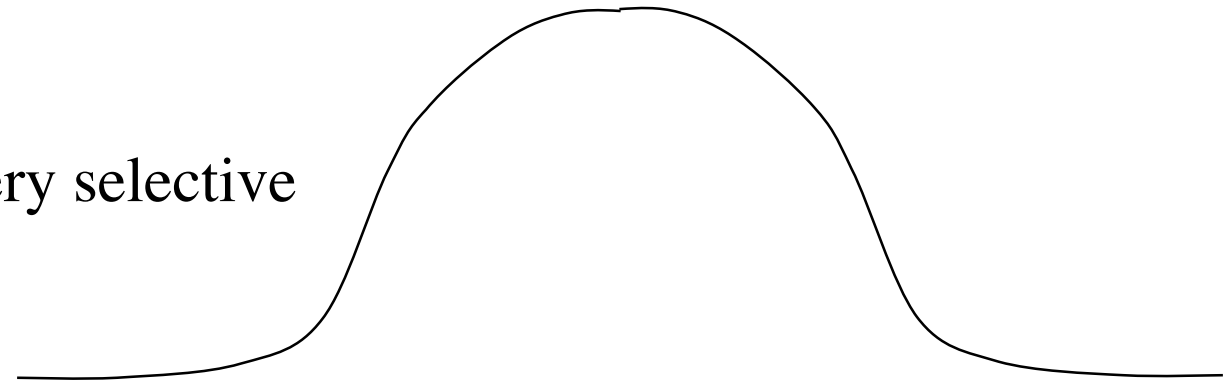


Spread = 1/Selectivity

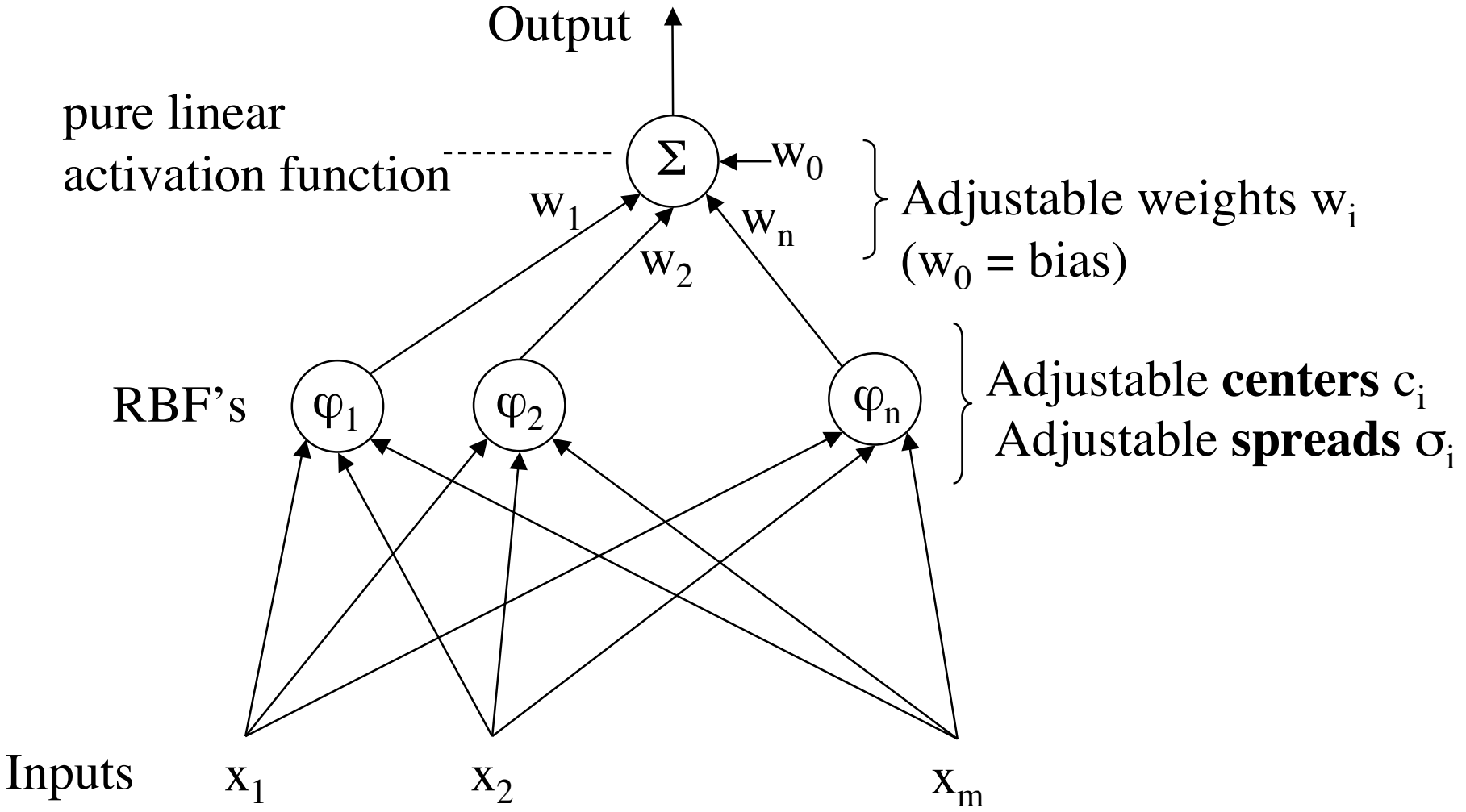
Small Spread, very selective



Large Spread, not very selective

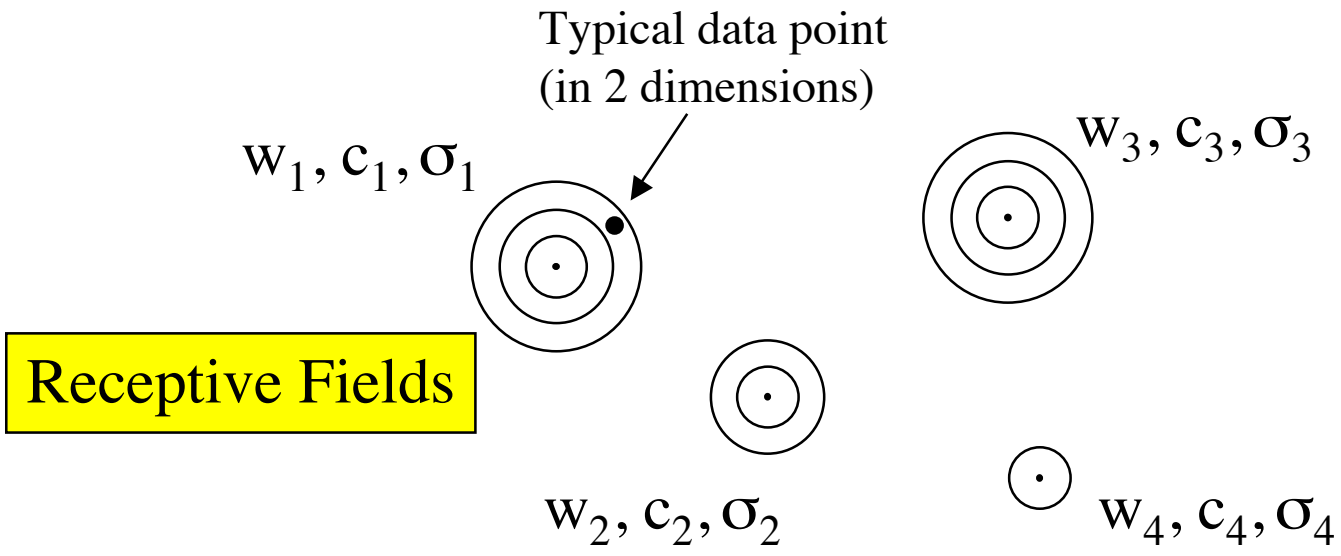


Radial Basis Function (RBF) Network: 2 Layers Only!



Radial Basis Function (RBF) Network in terms of Receptive Fields

Output = $\Sigma w_i \varphi_i(\mathbf{x})$ where $\mathbf{x}$ is the input vector



Determination of Parameters

- Given a set of data, the weights, centers, and spreads need to be determined for the best fit.

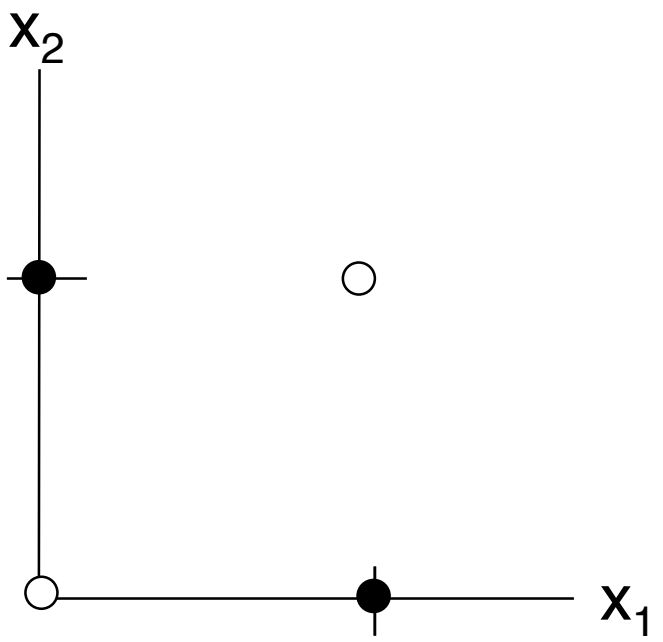
- Approaches:
  - Solving for all parameters
  - Determining centers and spreads by clustering, then training weights
  - Training for centers, spreads, and weights

Determination of Parameters

- The approaches mentioned assume a **specified number** of hidden-layer nodes.
- Another approach is to **add nodes successively**, until the approximation is good.
- In the **limit**, this might be **one node per training pattern**.

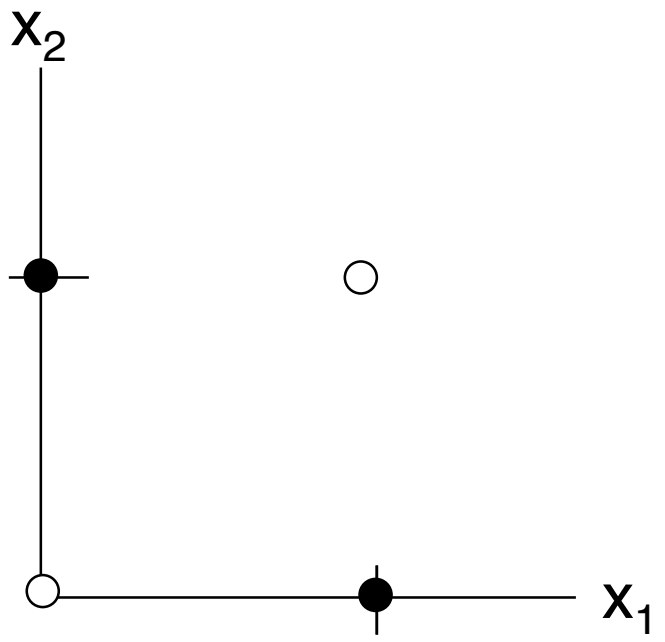
Example: xor

- How to choose parameters to realize xor with 2 unit RBF?
- Since output of an RBF is *linear*, would need to add a **limiter** to the general RBF.

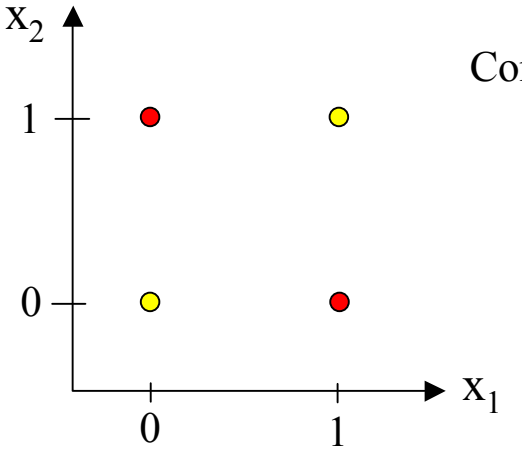


Example: xor

- Choose centers at (1, 0), and (1, 0). Choose spreads as, say 0.1, find weights.



XOR using RBF



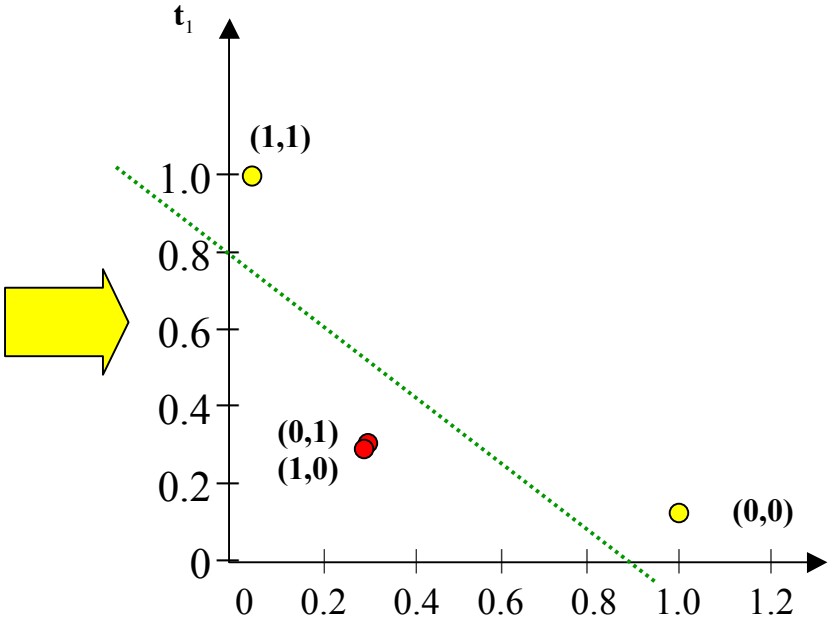
Consider the nonlinear functions to map the input vector **x** to the φ_1- φ_2 space

$\mathbf{x}=[x_1\ x_2]$

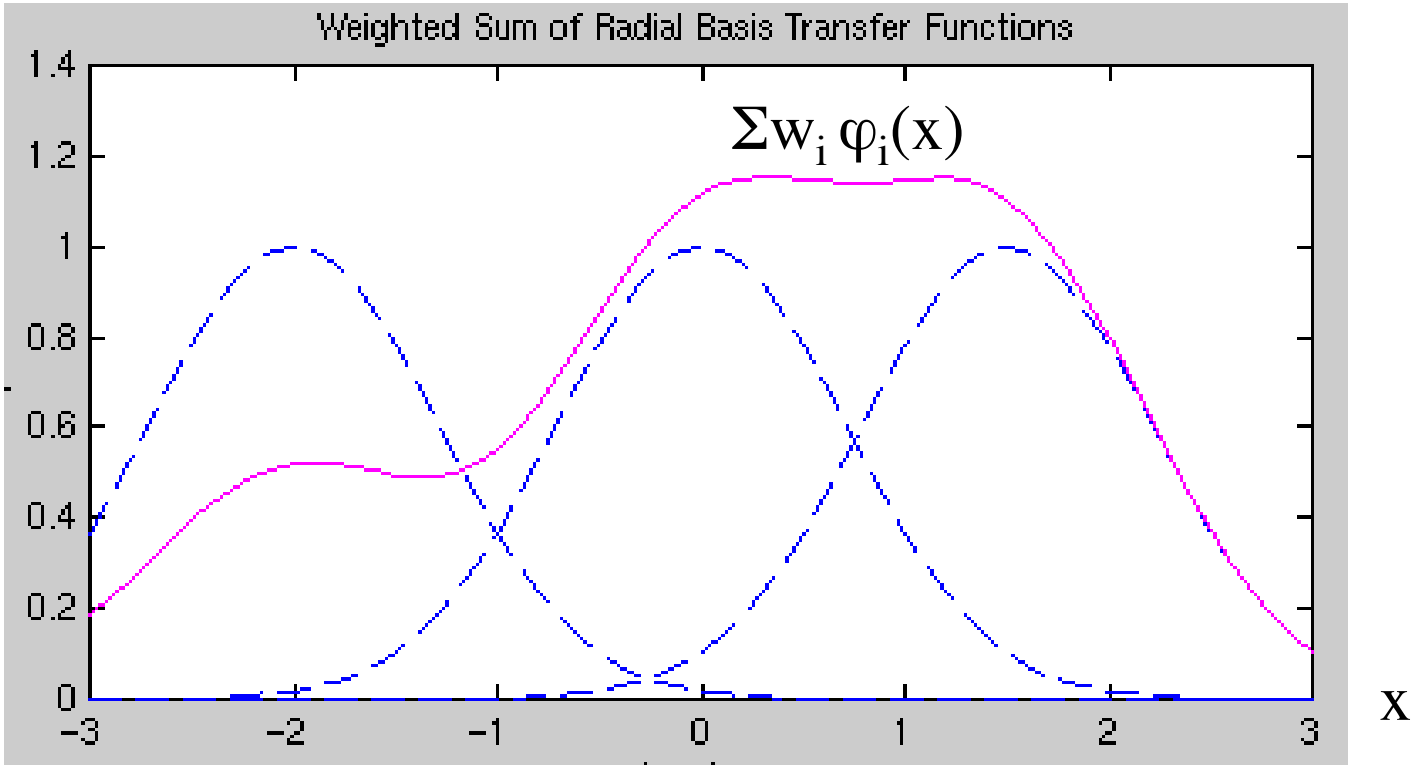
$$\varphi_1(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{t}_1\|^2} \qquad \mathbf{t}_1 = [1\ \ 1]^T$$

$$\varphi_2(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{t}_2\|^2} \qquad \mathbf{t}_2 = [0\ \ 0]^T$$

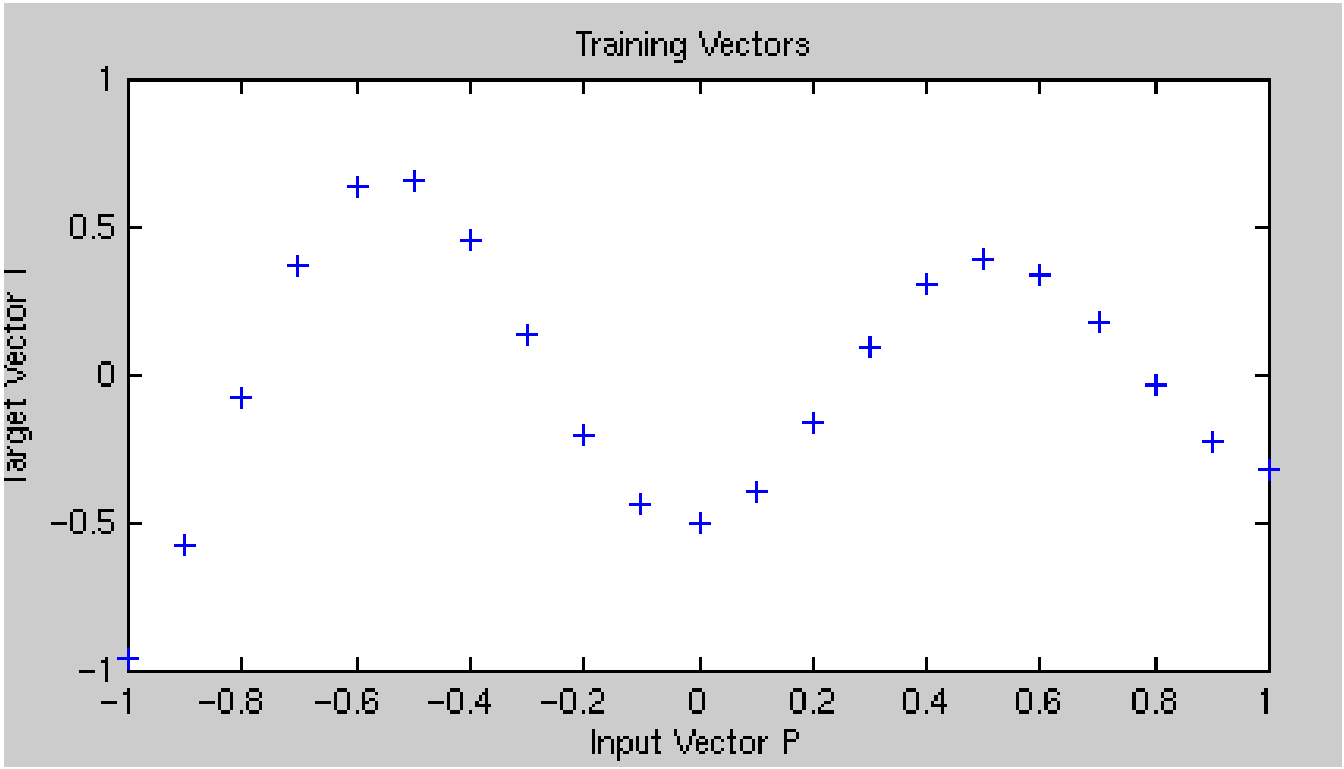
Input x	$\varphi_1(\mathbf{x})$	$\varphi_2(\mathbf{x})$
(1,1)	1	0.1353
(0,1)	0.3678	0.3678
(1,0)	0.3678	0.3678
(0,0)	0.1353	1



Example: 1D Function Approx. using RBF

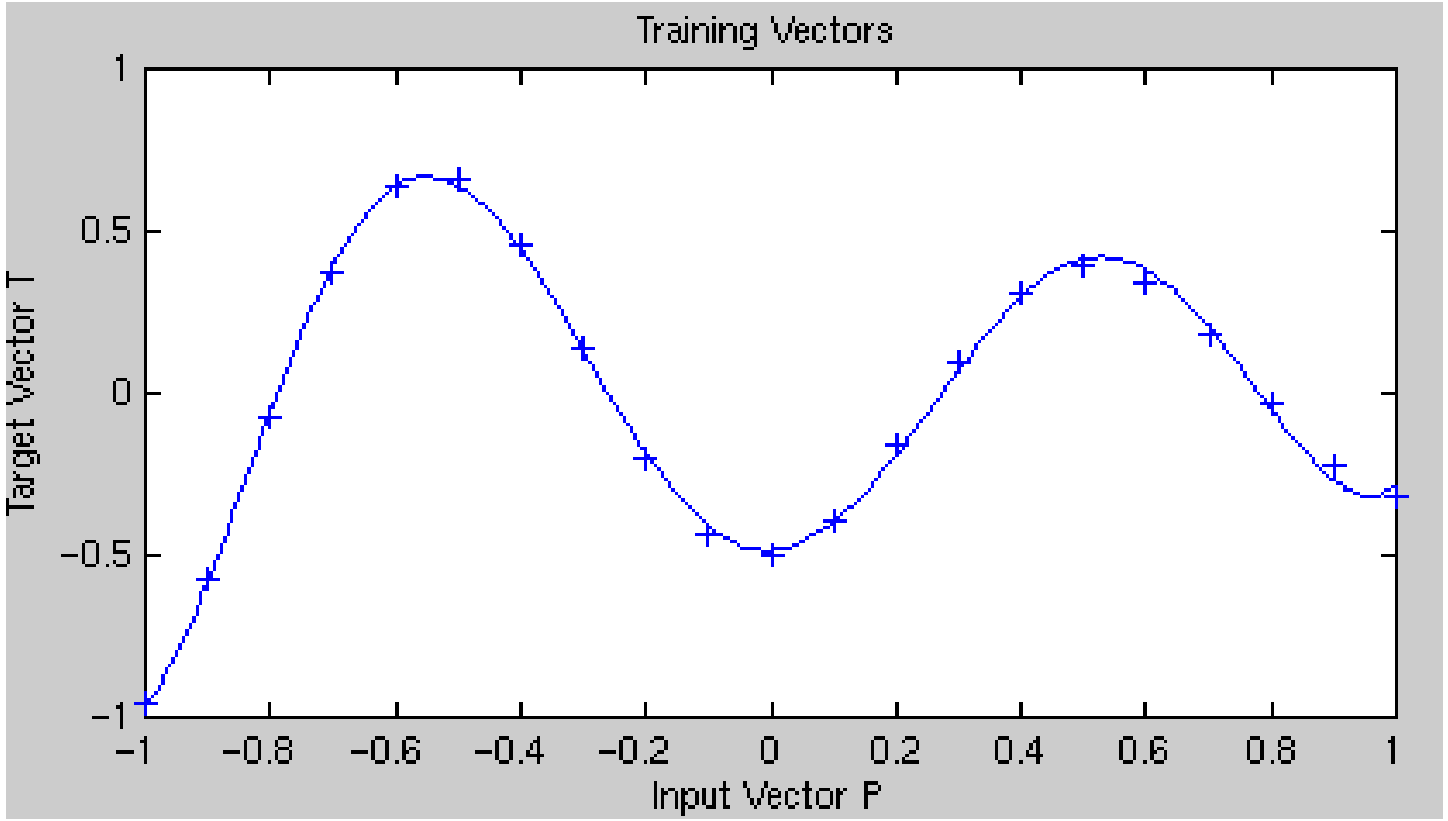


Matlab demorb1 input



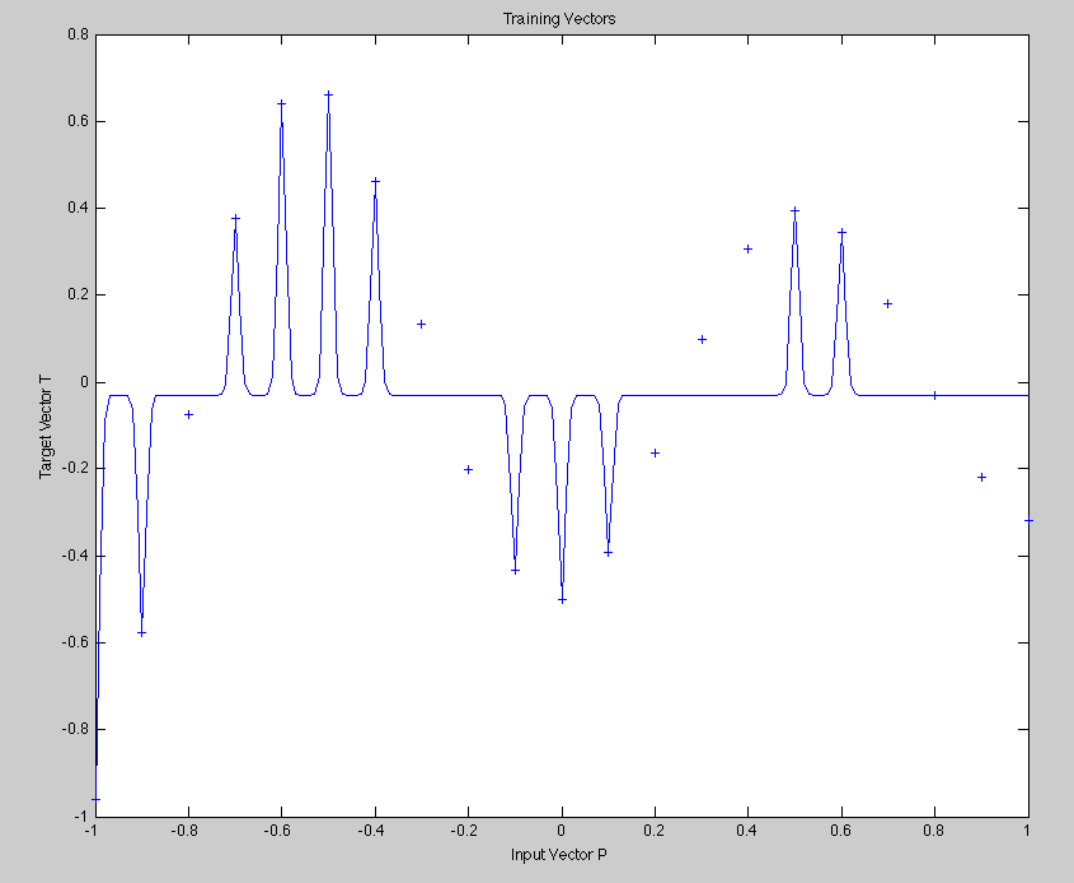
Matlab demorb1 function

Here all spreads = 1.



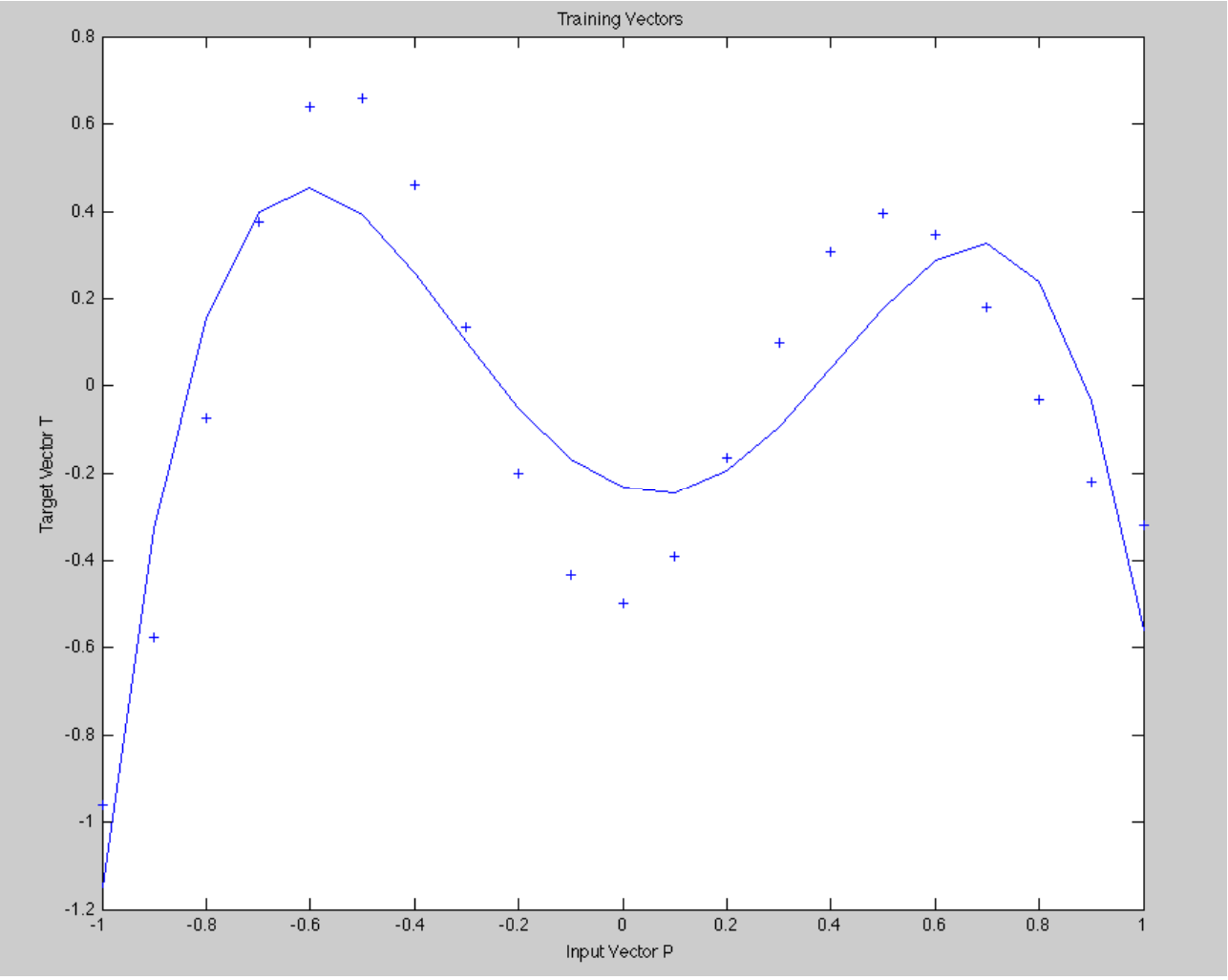
demorb3: spreads too small, poor network generalization

Here all spreads = 0.01 (vs. 1.0 in previous case).



demorb4: spreads too large, network over-generalization

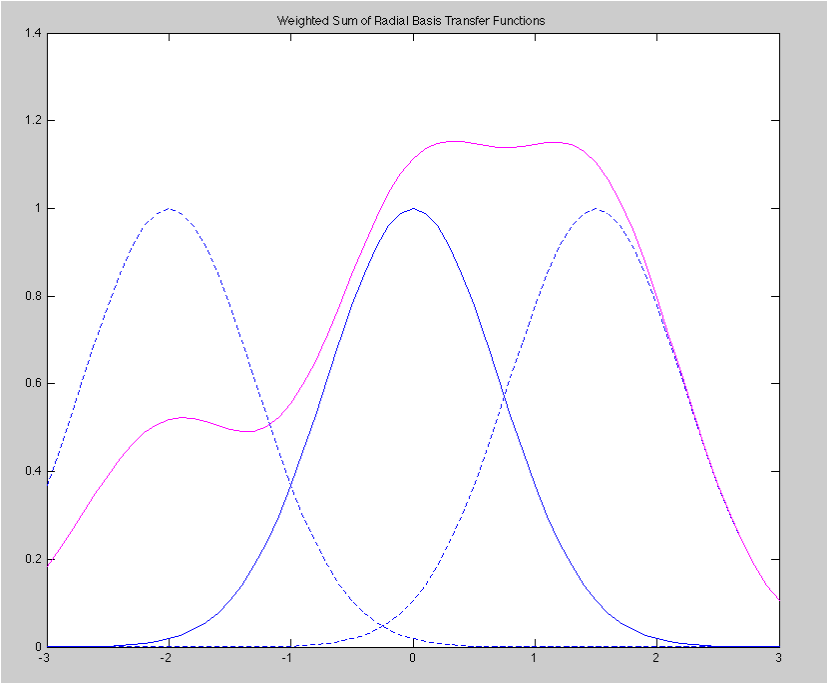
Here all spreads = 100.



Matlab code for the example in demorb1

```
P = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609 ...
      .1336 -.2013 -.4344 -.5000 -.3930 -.1647  .0988 ...
      .3072  .3960  .3449  .1816 -.0312 -.2189 -.3201];
```

```
p = -3:.1:3;
a = radbas(p);
a2 = radbas(p-1.5);
a3 = radbas(p+2);
a4 = a + a2*1 + a3*0.5;
plot(p,a,'b-',p,a2,'b--',
   p,a3,'b--',p,a4,'m-')
```



Bias-Variance Dilemma

- “Bias-variance dilemma” applies to the choice of spreads.

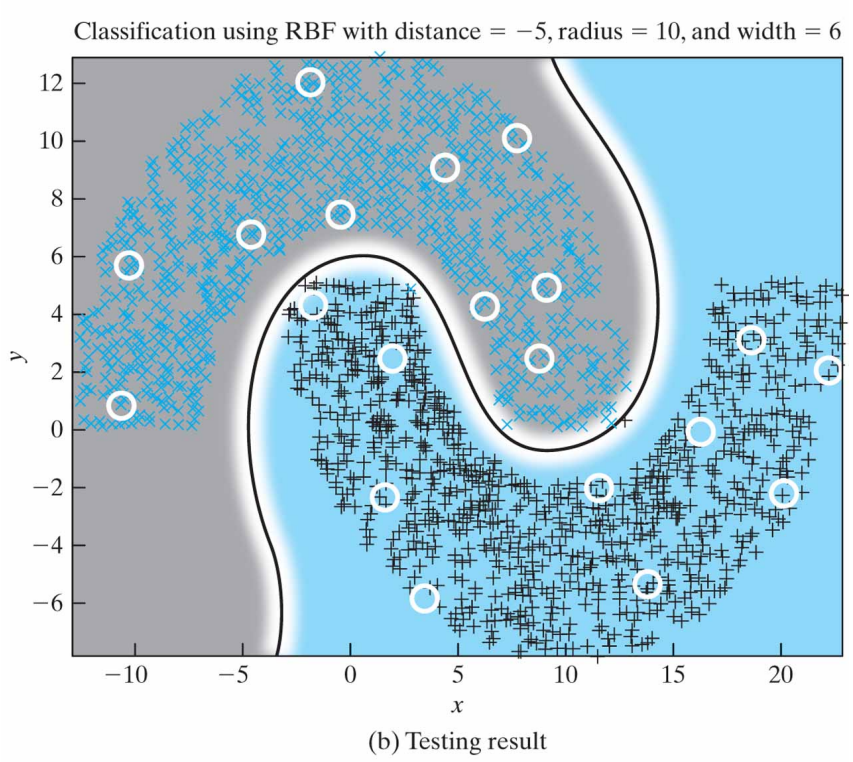
- ref. Neural and Adaptive Systems, Jose C. Principe, Neil R. Euliano, Curt Lefebvre

RBF Properties

- RBF networks tend to have good **interpolation** properties,
- but not as good **extrapolation** properties as MLP's.
- It seems pretty obvious why.
- For extrapolation, using a given number of neurons, an MLP could be a much better fit.

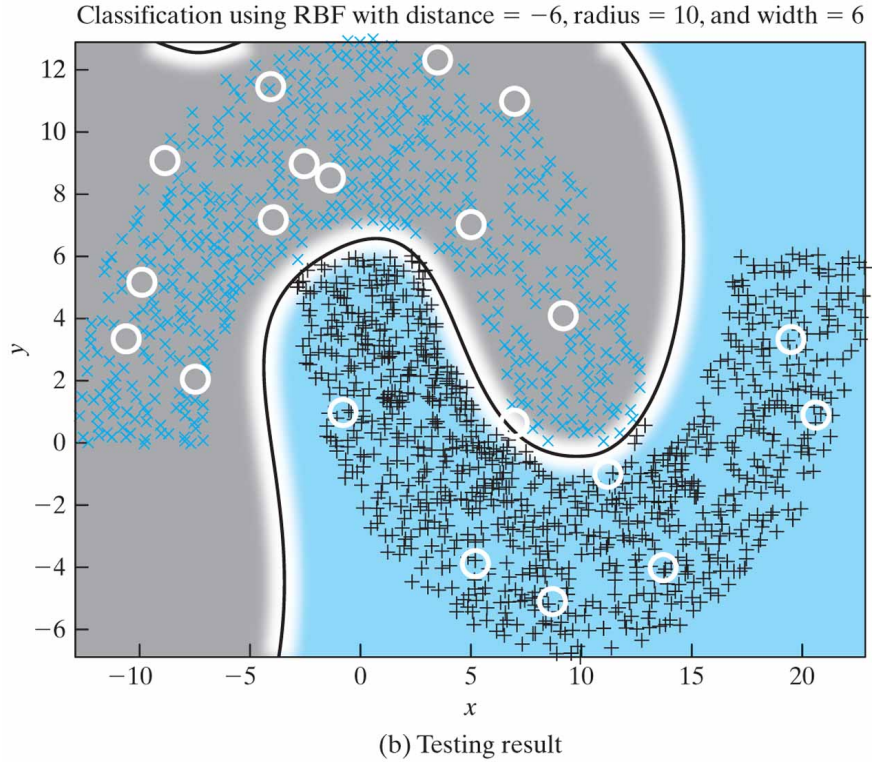
“2-Moons” Problems using RBF

distance = -5 problem



(b) Testing result

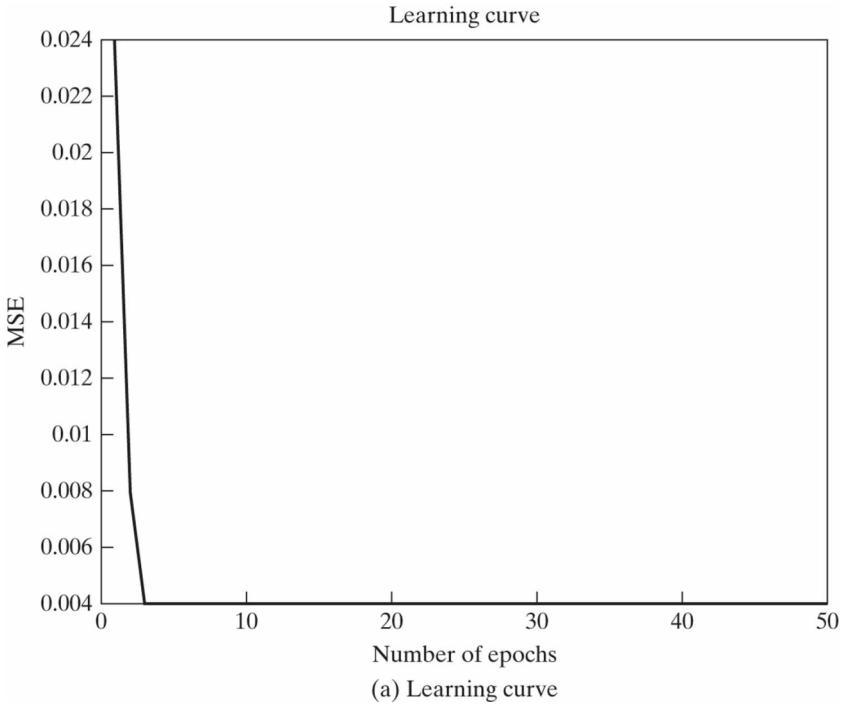
distance = -6 problem



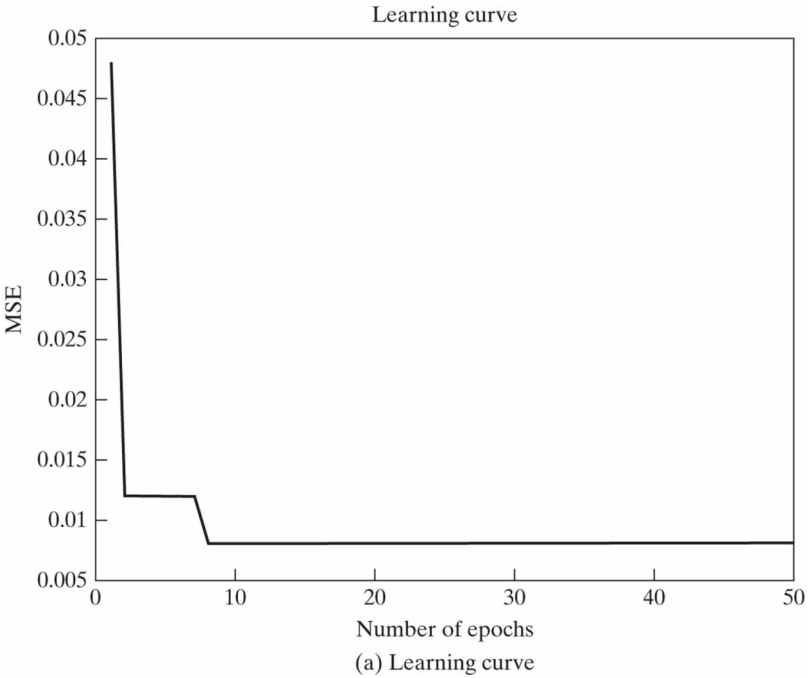
(b) Testing result

“2-Moons” Problems using RBF: Supervised Training of Weights

distance = -5 problem



distance = -6 problem



Training-Performance and Universality

- With proper setup, RBFNs can train in time **orders of magnitude faster** than backpropagation.

- RBFNs enjoy the same **universal approximation** properties as MLPs: given sufficient neurons, any reasonable function can be approximated (with just 2 layers).

Training Approach for RBFNs

- Haykin, Editions 1-2, section 5.13, gave update formulas for simultaneously **training** weights, centers, and spreads iteratively using **gradient descent**. This has become problem 5.8 in Edition 3.

Training for Weights (Haykin)

Error = $\varepsilon = \frac{1}{2}\sum_{j=1}^{N} e_j^2$ (j is the sample index)

$$e_j = d_j - \sum_{i=1}^{M} w_k \varphi\left(\left\|\mathbf{x}_j - \mathbf{t}_i\right\|\right)$$ (i is the neuron index in the hidden layer)

$$G\left(\left\|\mathbf{x}_j - \mathbf{t}_i\right\|_C\right) = \varphi\left(\left\|\mathbf{x}_j - \mathbf{t}_i\right\|\right)$$

G is for “Green’s Function”

1. *Linear weights* (output layer)

(j is the sample index, i is the weight index)

$$\frac{\partial \mathcal{E}(n)}{\partial w_i(n)} = \sum_{j=1}^{N} e_j(n) G(\|\mathbf{x}_j - \mathbf{t}_i(n)\|_{C_i})$$

$$w_i(n+1) = w_i(n) - \eta_1 \frac{\partial \mathcal{E}(n)}{\partial w_i(n)}, \qquad i = 1, 2, \ldots, m_1$$

2. *Positions of centers* (hidden layer)

$$\frac{\partial \mathcal{E}(n)}{\partial \mathbf{t}_i(n)} = 2 w_i(n) \sum_{j=1}^{N} e_j(n) G'(\|\mathbf{x}_j - \mathbf{t}_i(n)\|_{C_i}) \boldsymbol{\Sigma}_i^{-1} [\mathbf{x}_j - \mathbf{t}_i(n)]$$

$$\mathbf{t}_i(n+1) = \mathbf{t}_i(n) - \eta_2 \frac{\partial \mathcal{E}(n)}{\partial \mathbf{t}_i(n)}, \qquad i = 1, 2, \ldots, m_1$$

3. *Spreads of centers* (hidden layer)

$$\frac{\partial \mathcal{E}(n)}{\partial \boldsymbol{\Sigma}_i^{-1}(n)} = - \; w_i(n) \sum_{j=1}^{N} e_j(n) G'(\|\mathbf{x}_j - \mathbf{t}_i(n)\|_{C_i}) \mathbf{Q}_{ji}(n)$$

$$\mathbf{Q}_{ji}(n) = [\mathbf{x}_j - \mathbf{t}_i(n)][\mathbf{x}_j - \mathbf{t}_i(n)]^T$$

$$\boldsymbol{\Sigma}_i^{-1}(n+1) = \boldsymbol{\Sigma}_i^{-1}(n) - \eta_3 \frac{\partial \mathcal{E}(n)}{\partial \boldsymbol{\Sigma}_i^{-1}(n)}$$

G’ is the derivative of the “Green’s Function”

Some Wisdom on Training RBFN's

- Supervised training for centers and spreads is reportedly very slow.
- Thus some have taken the approach of computing these parameters by other means and just training for the weights (at worst).

A Solving Approach for RBF

- Suppose we are willing to use **one neuron per training sample point**.
- Choose the N data points themselves as centers.
- Assume the spreads are given.
- It then only remains to **find the weights**.
- Define $\varphi_{ji} = \varphi(\| x_i - x_j \|)$ where φ is the radial basis function, x_i, x_j are training samples.
- The matrix Φ of values φ_{ji} is called the **interpolation matrix**.

Solving Approach for RBF

- The **interpolation matrix** has the property that

  $$\Phi \mathbf{w} = \mathbf{d}$$

  where
  - **w** is the weight vector
  - **d** is the desired output vector over all training samples (since the samples are both data points and centers).

- If Φ is non-singular, then we can **solve** for weights as

  $$\mathbf{w} = \Phi^{-1}\mathbf{d}$$

Solving Approach for RBFNs

- **Micchelli's Theorem** says that if the points x_i are distinct, then the Φ matrix ***will*** be non-singular (it is square, by construction).

- Ref: Mhaskar and Micchelli, *Approximation by superposition of sigmoidal and radial basis functions*, Advances in Applied Mathematics, 13, 350-373, 1992.

Non-Square Case

- Having one RBF per training point may be too costly.
- It may also cause over-fitting.

Training in the Presence of Noise

- Noise in the training set can be good; it can make the resulting network, which has learned to “average” noise in, more **robust**.
- However, with too many neurons, a network can **over-train** to “learn the noise”.

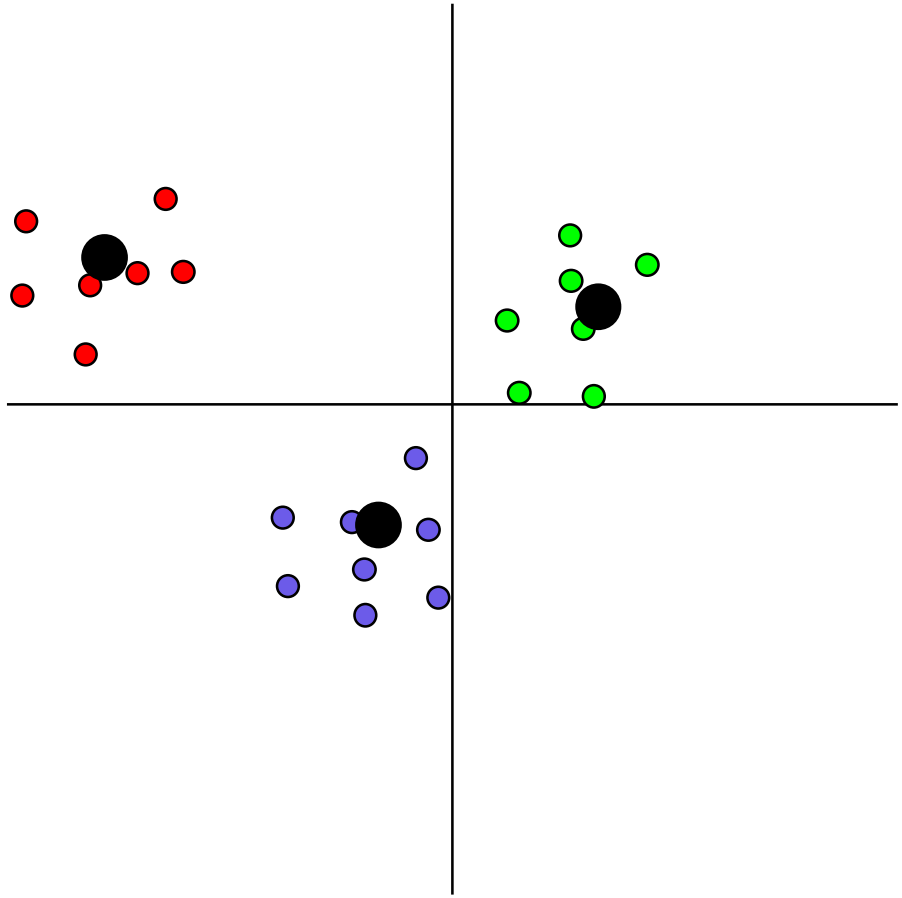
Regularization by Weight Decay

- Can use the **Weight Decay** method to **prune** an RBF:
  - At each update, a small amount is **deducted** from each weight.
  - Weights that are constantly being updated will end up with a non-0 value, while others will go to 0 and can be **eliminated**.
  - The resulting network is less trained to the noise.
- Weight Decay is a form of "**regularization**"

Selecting Centers by Clustering

- One center per training sample may be over-kill.
- There are ways to select centers as representatives among **clusters**, given a **fixed** number of representatives.
- We will give and example, and discuss these further under “unsupervised learning” and competitive methods.

Selecting Centers by Clustering



“k-means clustering”

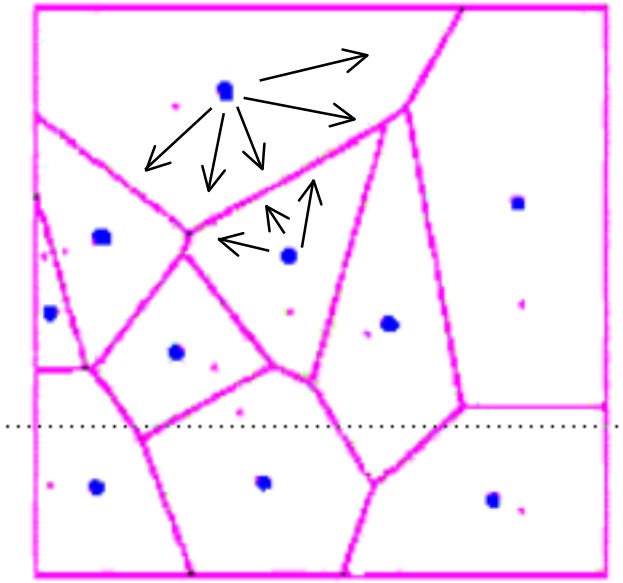
(MacQueen 1967)

- This determines which points belong to which clusters, as well as the centers of those clusters.
- The desired **number k** of clusters is specified.
- Initialize **k centers**, e.g. by choosing them to be k distinct data points.
- Repeat
  - For each data point, determine which center is closest. This determines each point’s **cluster** for the current iteration.
  - Compute the centroid (**mean**) of the points in each cluster. Make this the centers for the next iteration.
- until centers don’t differ appreciably from their previous value.

Voronoi Tessellation (aside)

- The Voronoi Tessellation is a way to **visualize** how the centers divide the space of possible data points.

  A region in the tessellation consists of all points in the space that are **closest to a given center**.



“k-means clustering”

- Tries to optimize the SSE of the difference between points and the center of their clusters.

$$E = \sum_{j=1}^{n_c} \sum_{i=1}^{N} \left(\| v_i - c_j \| \right)^2$$

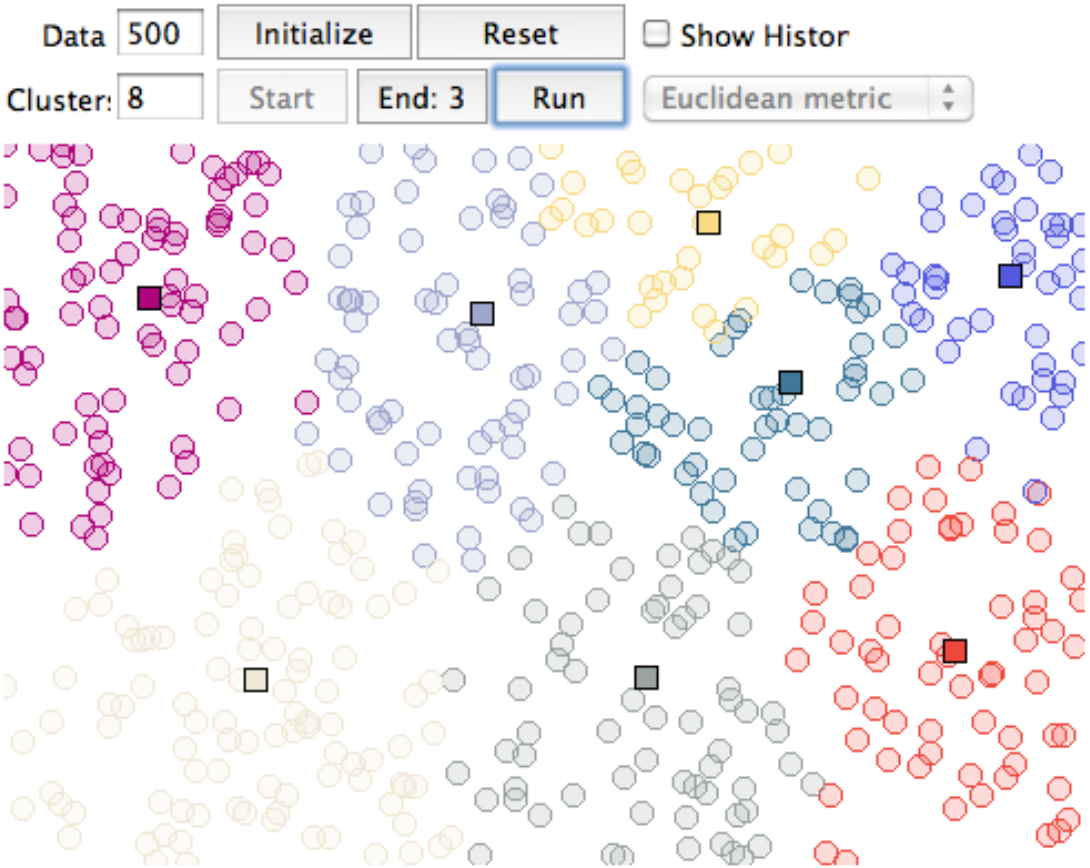
- This is a heuristic procedure, and is subject to the usual **local minima** pitfalls.

- However, it is used quite often.

cf. http://www.journal.au.edu/ijcim/august98/improve_3.html

Improving the Performance of K-Means Clustering Algorithm to Position the Centres of RBF Network

k-means demo



http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

Computing weights using fewer centers than points

- We can't invert the Φ matrix in this case (recall $\Phi\mathbf{w} = \mathbf{d}$).

- We can find the weight values that minimize the error $\Phi\mathbf{w} - \mathbf{d}$ using the "pseudo inverse" technique for least squares :

$$\mathbf{w} = \underbrace{(\Phi^T \Phi)^{-1} \Phi^T}_{\text{pseudo-inverse of } \Phi} \mathbf{d}$$

Setting spreads

- Once centers are known, spreads can be set, e.g. by selecting the **average distance** between center and the c closest points in the cluster (e.g. c = 5).

Example: matlab newrb

```
NEWRB(PR,T,GOAL,SPREAD,MN,DF) takes these arguments,
  P      - RxQ matrix of Q input vectors.
  T      - SxQ matrix of Q target class vectors.
  GOAL   - Mean squared error goal, default = 0.0.
  SPREAD - Spread of radial basis functions, default = 1.0.
  MN     - Maximum number of neurons, default is Q.
and returns a new radial basis network.

The larger that SPREAD is the smoother the function approximation
will be.  Too large a spread means a lot of neurons will be
required to fit a fast changing function.  Too small a spread
means many neurons will be required to fit a smooth function,
and the network may not generalize well.  Call NEWRB with
different spreads to find the best value for a given problem.
```

Method of newrb

Initially the RBF layer has no neurons. The following steps are repeated until the network's mean squared error falls below GOAL.

1) The network is simulated.

2) The input vector with the greatest error is found.

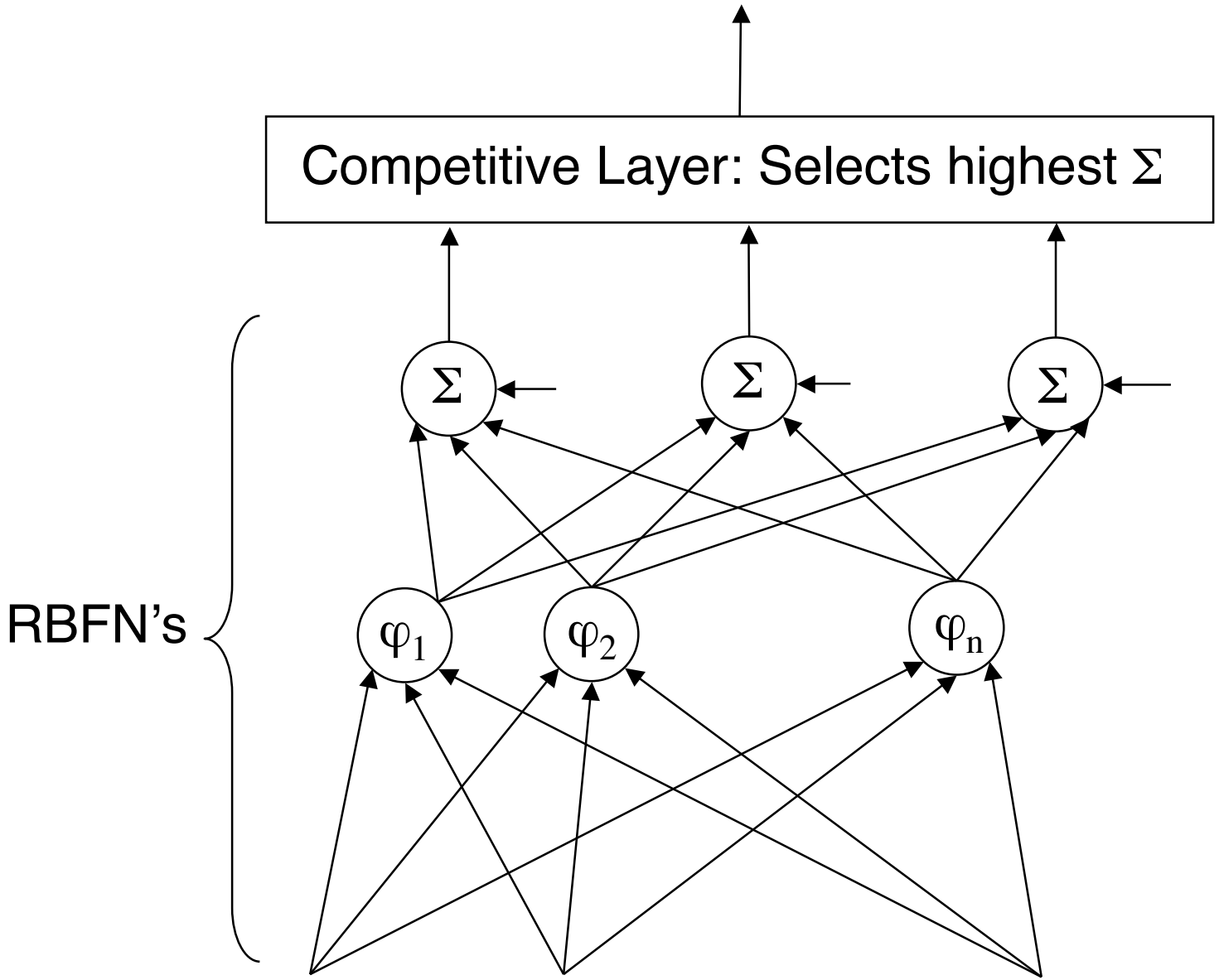
3) A RADBAS neuron is added with center equal to that vector.

4) The PURELIN layer weights are redesigned by solving a set of linear equations.

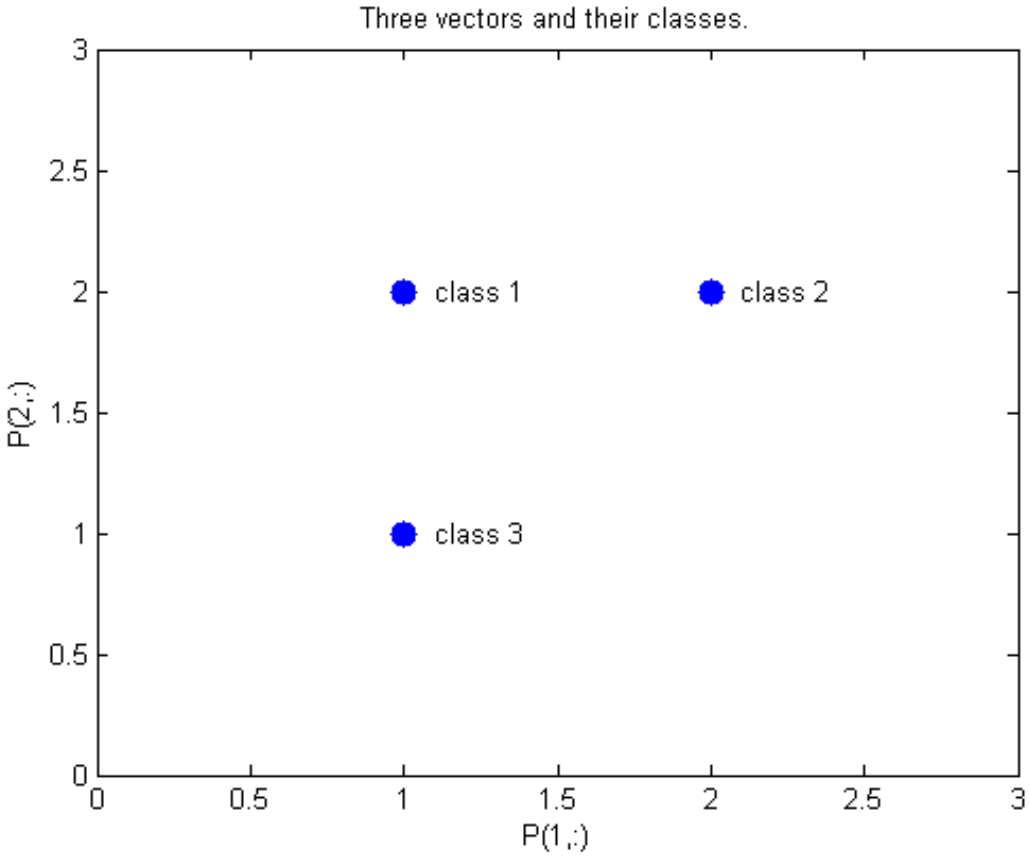
“Probabilistic” Neural Networks

- A “Probabilistic” Neural Network (PNN) is the name given to a radial-basis function network modified for **classification** purposes.

- The linear output layer is followed by a ***competitive*** layer which makes a **classification** based on the RBF unit with the **largest output**.


Competitive Layer: Selects highest Σ
RBFN's
Σ
Σ
Σ
φ1
φ2
φn

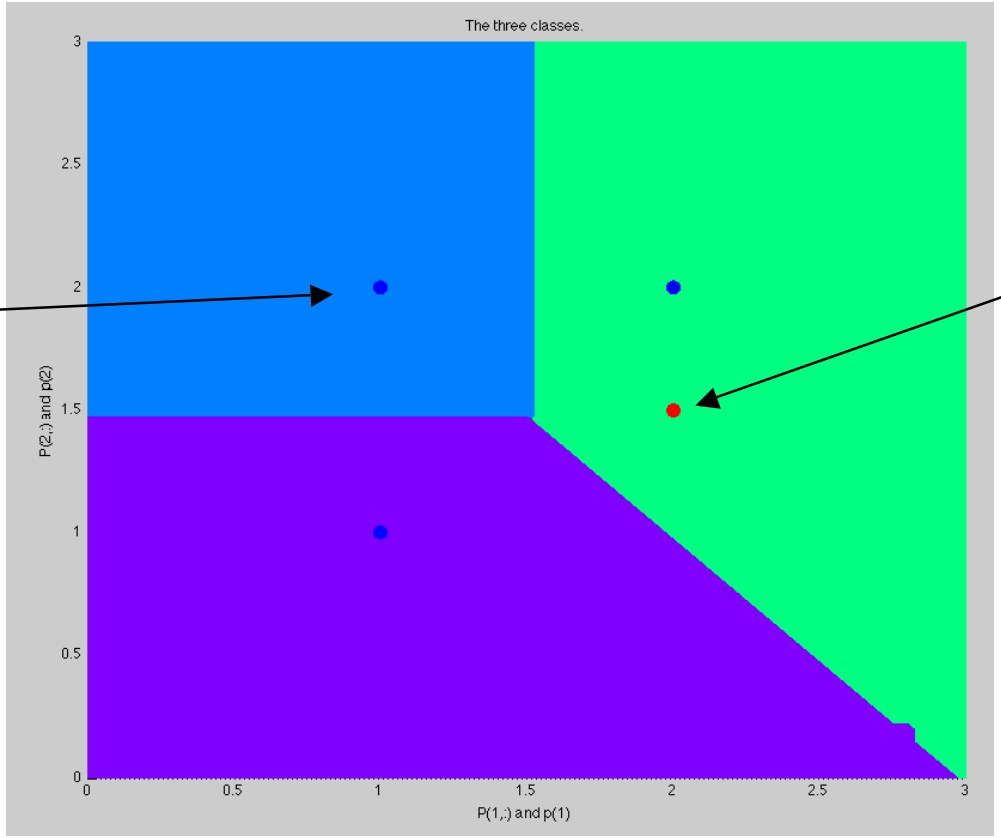
Matlab demopnn1



P = [1 2; 2 2; 1 1]'; T = ind2vec([1 2 3]);
net = newpnn(P,T,spread);

demopnn1 classification regions defined by training data

original training points (blue)



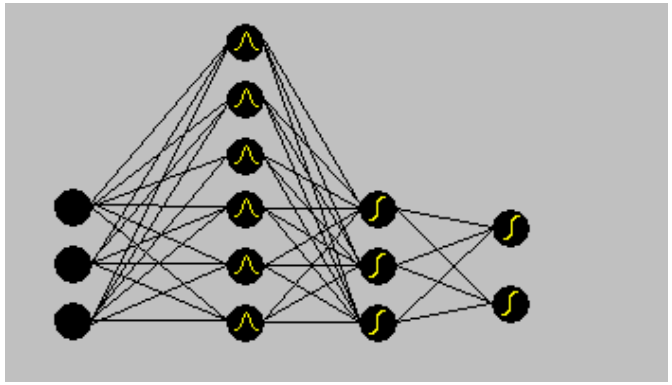
new test point (red), classified as 2

```
vec2ind(sim(net, P))
ans = 1     2     3
```

```
vec2ind(sim(net, [2; 1.5]))
ans = 2
```

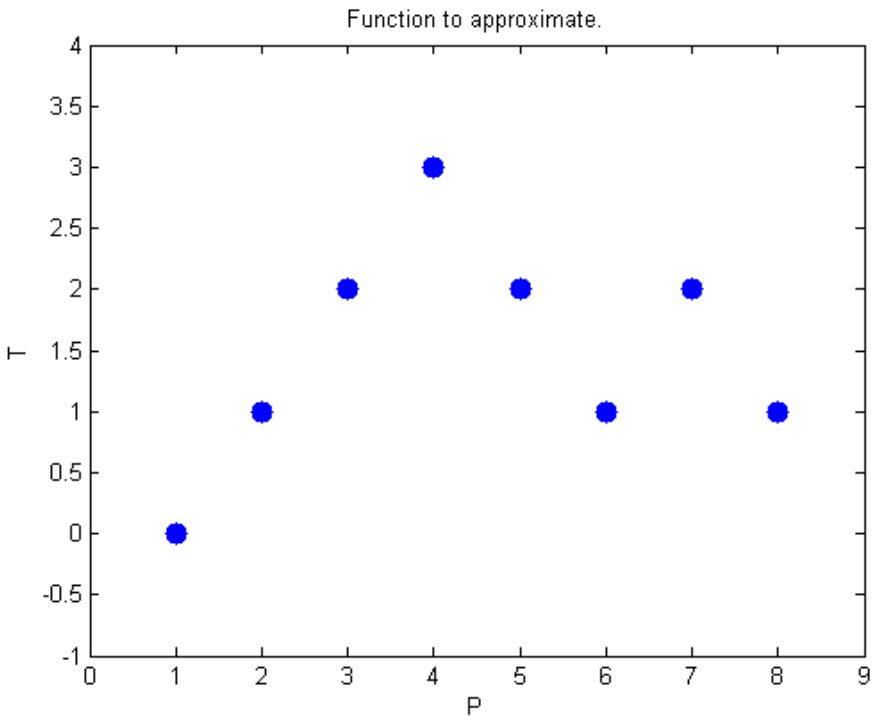
GRNN's

- **Generalized Regression Neural Networks** is another class that subsumes both RBFs and PNNs.



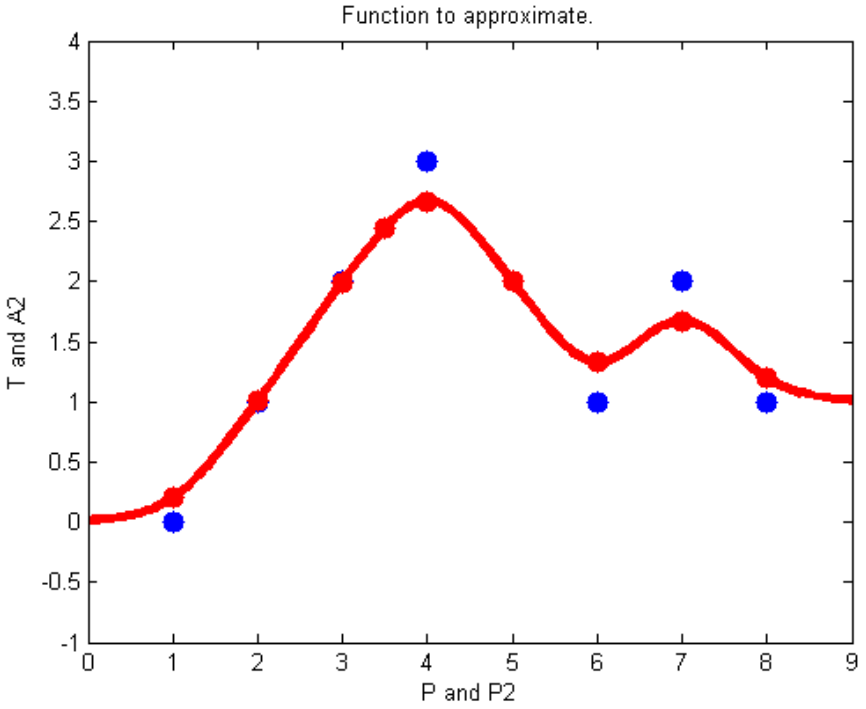
- They can be explained based on statistical estimation theory (Bayesian).

matlab demogrn1



Blue are Training Points

```
P = [1 2 3 4 5 6 7 8];
T = [0 1 2 3 2 1 2 1];
spread = 0.7;
net = newgrnn(P,T,spread);
A = sim(net,P);
```



Red is Network response to new Inputs P2

```
P2 = 0:.1:9;
A2 = sim(net,P2);
```

Related Topic to GRNN

- Support Vector Machines (SVMs) to be discussed

MLP vs RBF Case Studies
(source: Yampolskiy and Novikov, RIT)

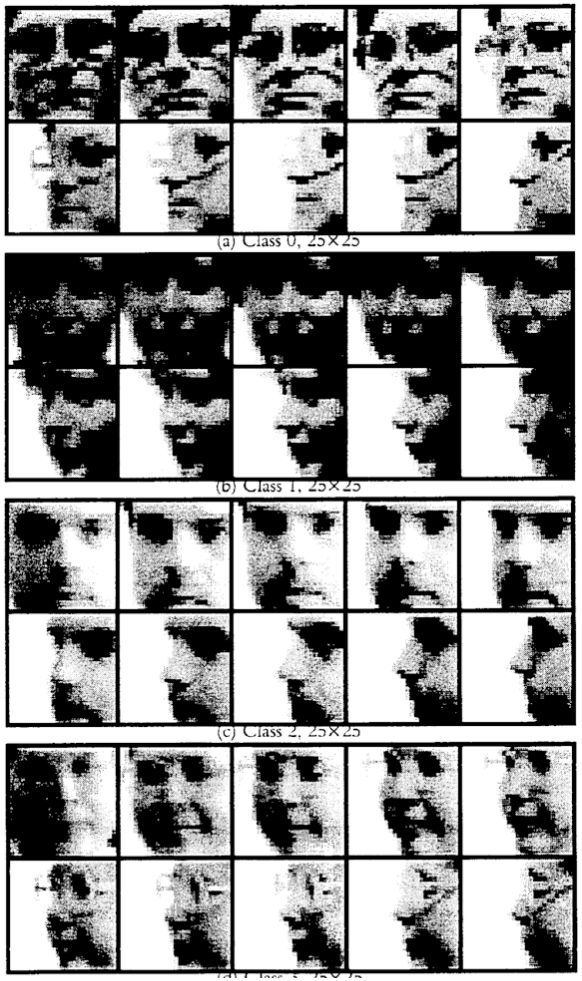
Source	Application	MLP	RBF
Dong	Satellite image classification		Faster runtime
Finan	Speaker recognition		More accurate, less sensitive to bad training data
Hawickhorst	Speech recognition		Faster training, better retention of generalization
Li	Surgical decision making	Fewer hidden nodes	Shorter training time, lower errors
Lu	Channel Equalization	Statistically insignificant differences	
Park	Nonlinear system identification		Better convergence to global min., less retraining time
Roppel	Odor recognition	Higher identification rates	

Face Recognition Case Study

(Powell, et al. at University of Sussex)

- **Database**
  - **100 images of 10 people** (8-bit grayscale, resolution 384 x 287)
  - for each individual, 10 images of head in different pose **from face-on to profile**
  - Designed to asses performance of **face recognition techniques** when pose variations occur

Sample Images (different angles)



Approach: Face-unit subnets

- Each of a set of RBF sub-networks is trained to **recognize a single person**.
- Training uses examples of images of the person to be recognized as positive evidence, together with selected confusable images of other people as negative evidence.

Network Architecture

- Input layer contains 25*25=625 inputs which represent the pixel intensities (normalized) of an image.
- Hidden layer contains p+a (pro+anti) neurons:
  - pro-neurons (receptors for positive evidence)
  - anti-neurons (receptors for negative evidence)
- Output layer contains two neurons:
  - One for the particular person.
  - One for all the others.

The output is “discarded” (discounted) if the absolute difference of the two output neurons is smaller than a **parameter R**.

Parameters

- **Centers**:
  - of a pro neuron: the corresponding positive example
  - of an **anti neuron**: the negative example which is **most similar to the corresponding pro neuron**, with respect to the Euclidean distance.
- **Spread**: average distance of the center vector from all other centers. If α, *h* hidden nodes, H total number of hidden nodes then:

$$\sigma_\alpha = \frac{1}{H\sqrt{2}} \sum_h \| t^\alpha - t^h \|$$

- **Weights**: determined using the **pseudo-inverse method**.
- A RBF network with 6 pro neurons, 12 anti neurons, and R equal to 0.3.
- **Results:**
  - Discarded 23 percent of the images of the test set.
  - Correctly classified 96 percent of the non-discarded images.