

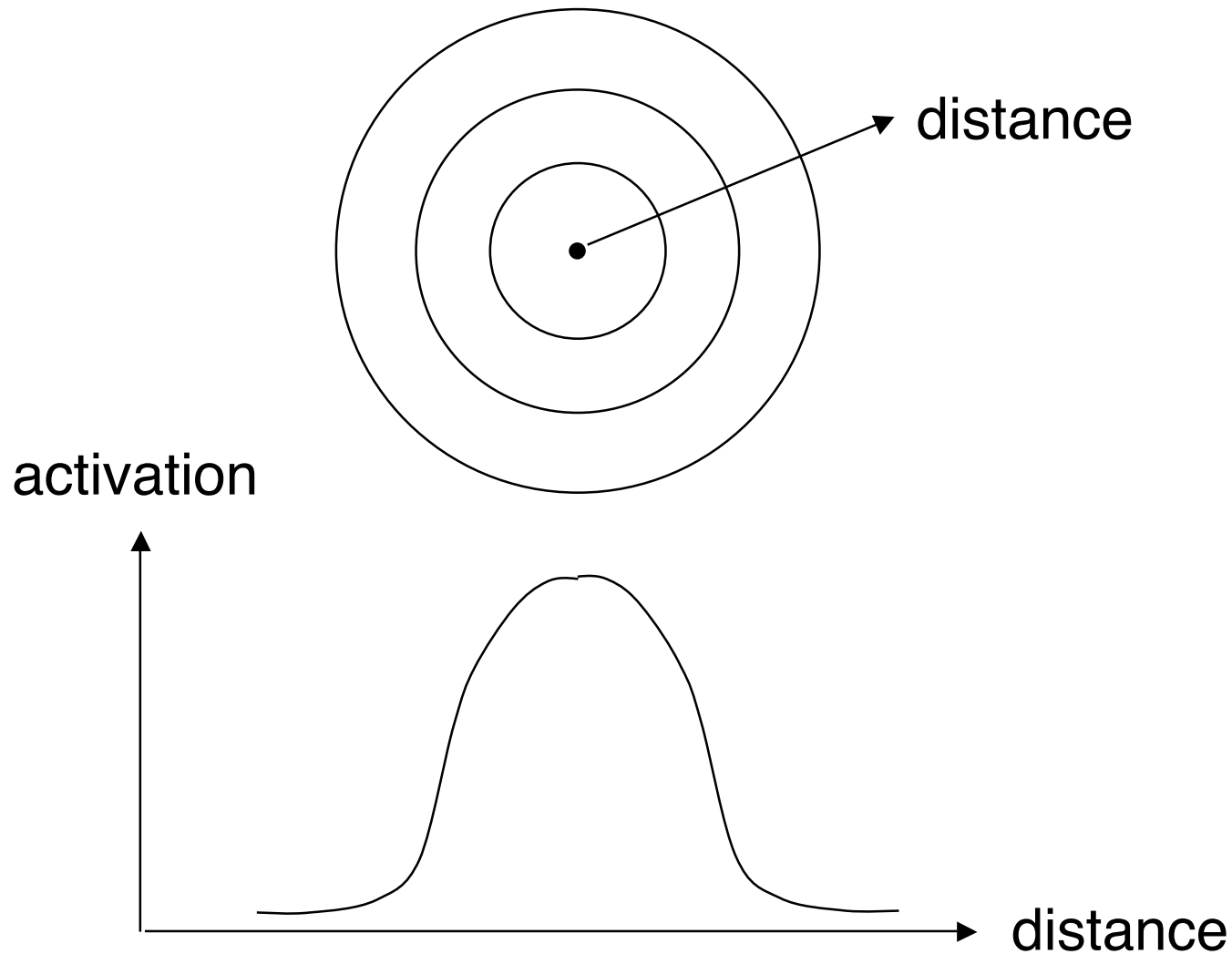


Radial Basis Function Networks

Radial Basis Functions

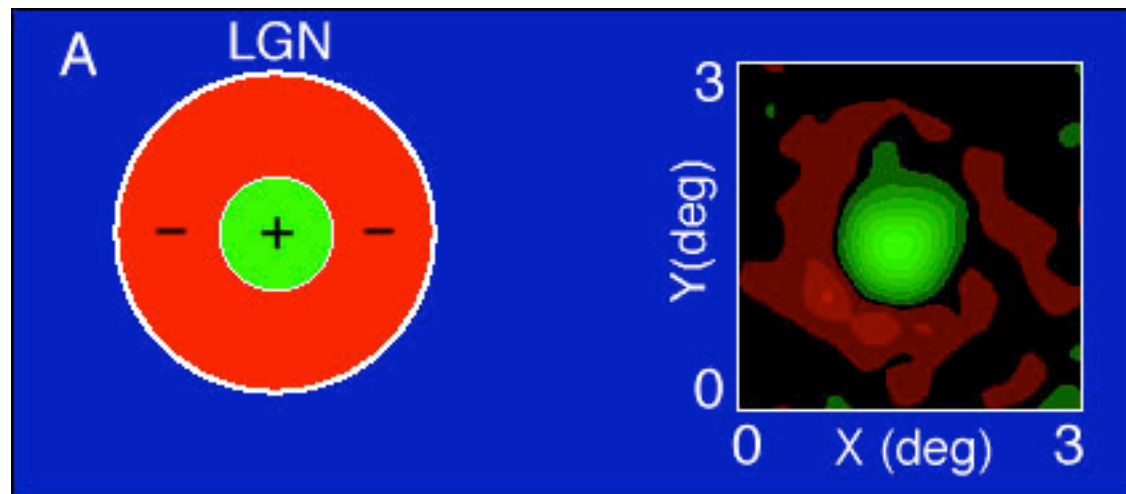
- In contrast to sigmoidal functions, radial basis functions have **radial symmetry** about a *center* in n-space ($n = \#$ of inputs).
- The **farther** from the center the input is, the **less** the activation.
- This models the “**on-center off-surround**” phenomenon found in certain **real neurons**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

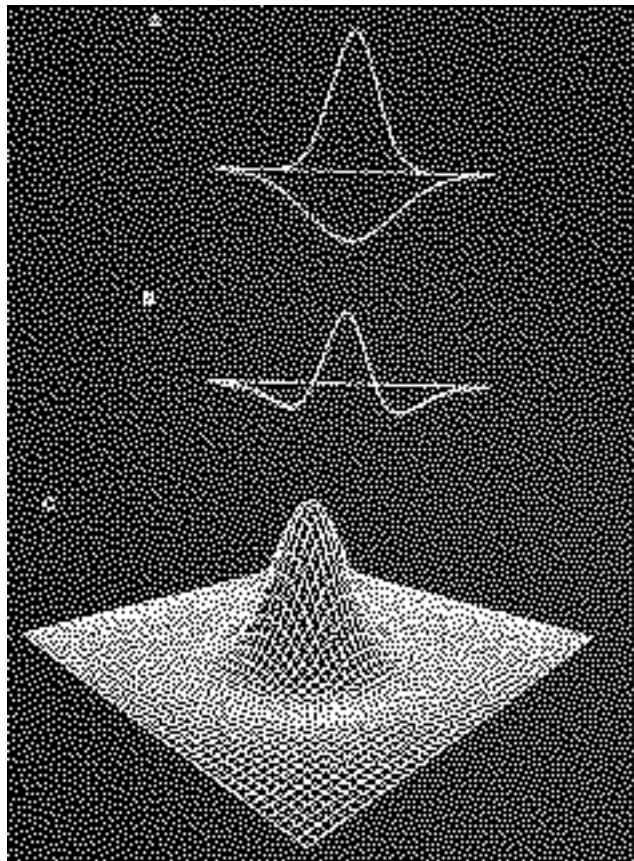
LGN description,

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.

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 ("sombbrero hat")



On-Center Behavior

- Also occurs in the ear (sensitivity to tones in the cochlear stereocilia cells).

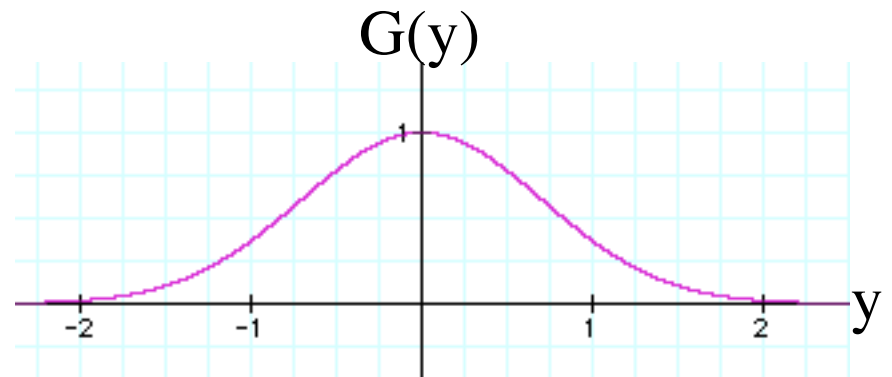
Modeling

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

where G is a decreasing function and \mathbf{c}_i is the **center**.

- Example: Gaussian:

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

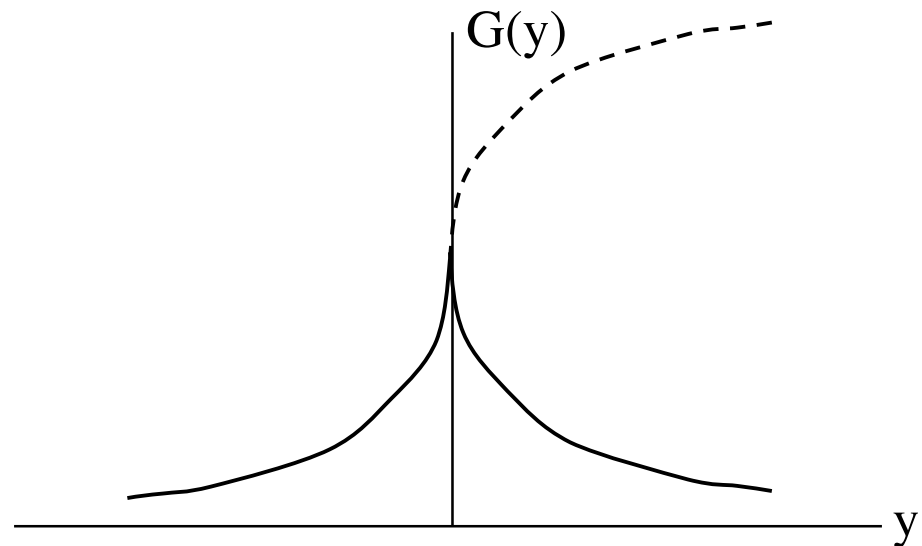


where σ is a parameter called the **spread**, which indicates the **selectivity** of the neuron.

Other RBF Examples

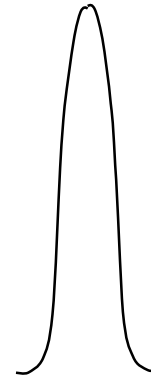
- $G(y) = 1/\sqrt{y^2 + \sigma^2}$
- $G(y) = 1/(1+\exp(ay^2))$

“reflected sigmoid”

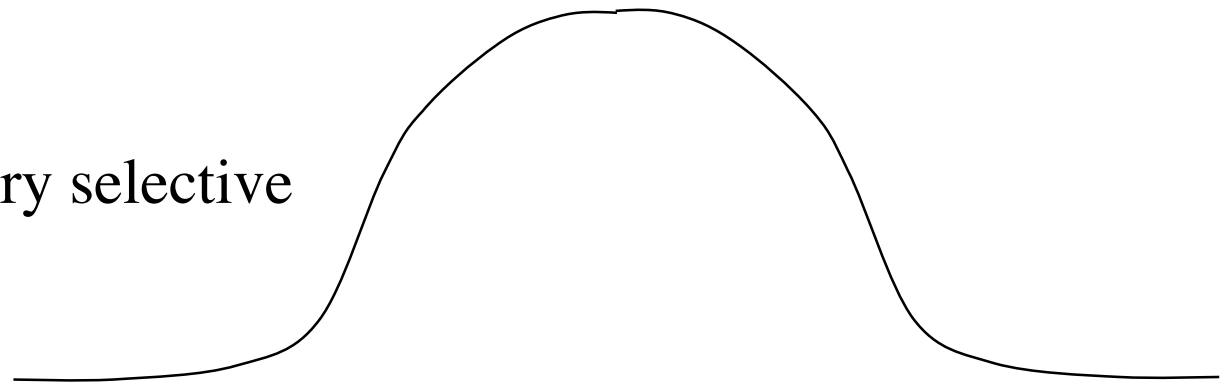


Spread = 1/Selectivity

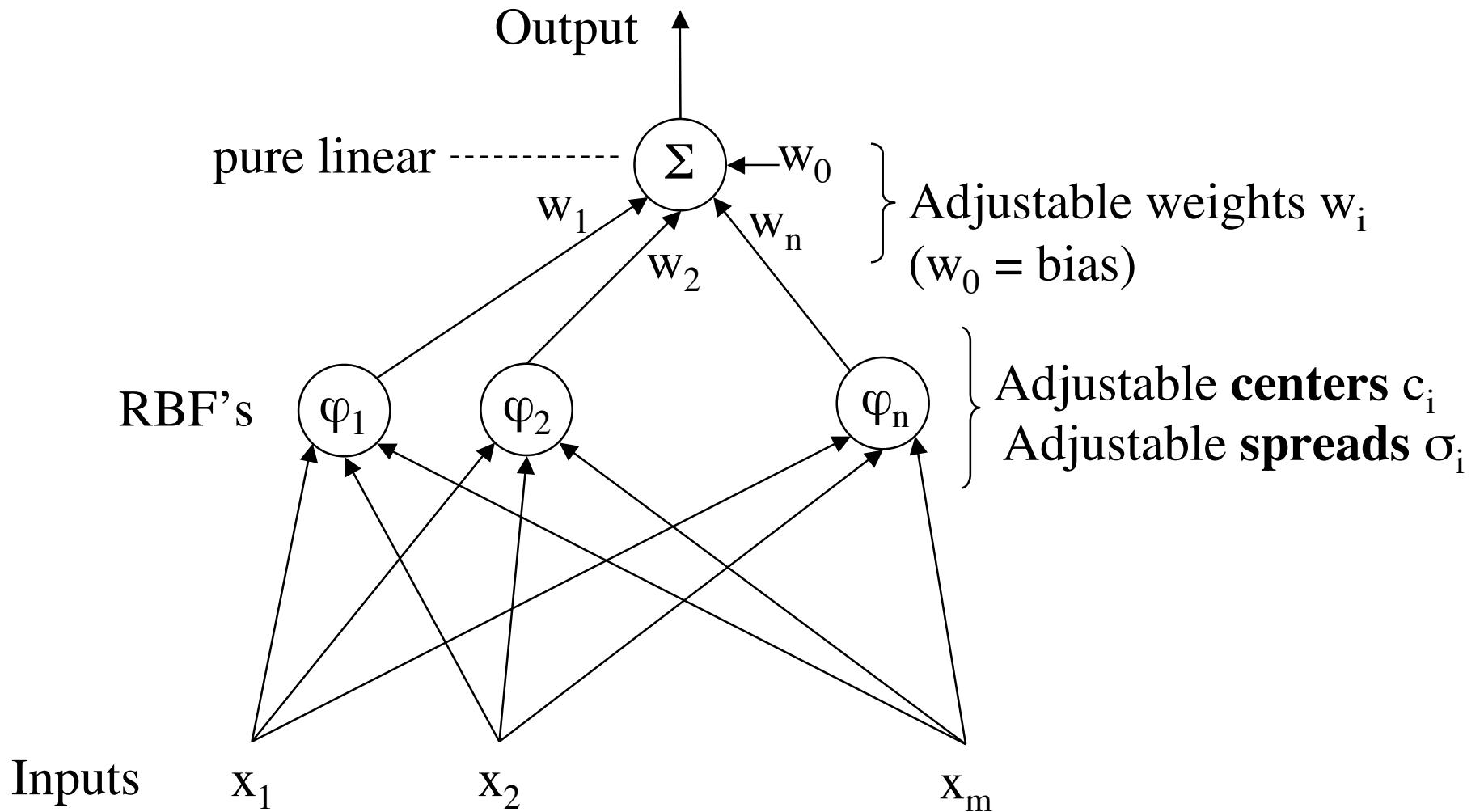
Small Spread, very selective



Large Spread, not very selective

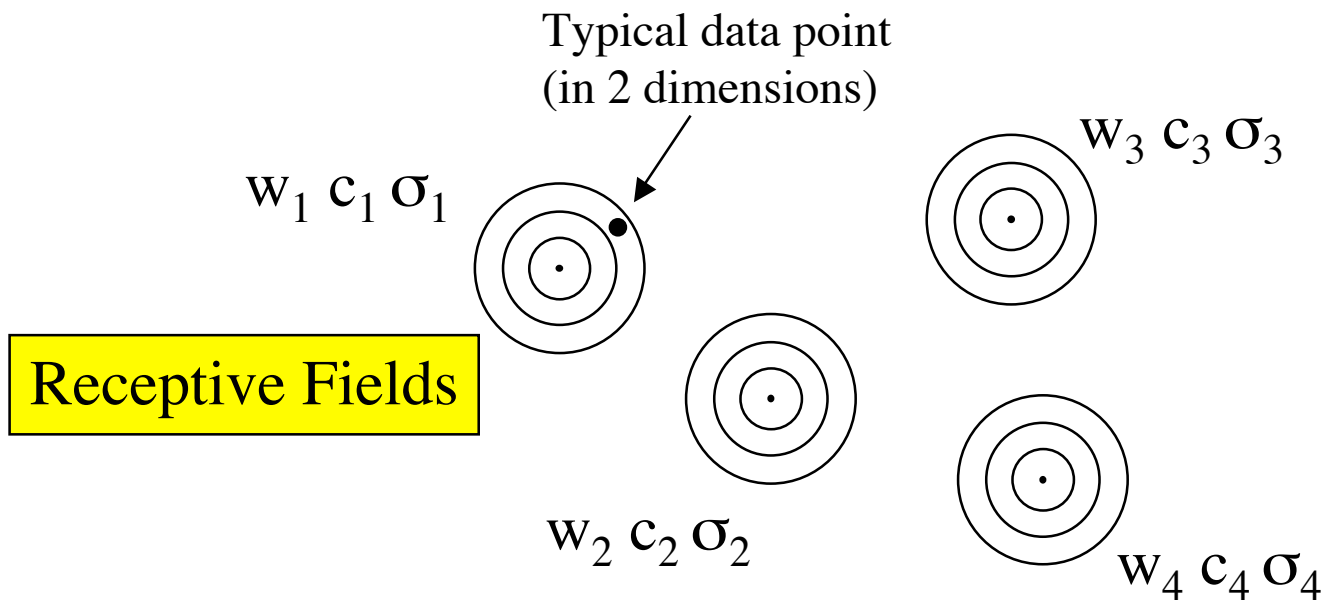


Radial Basis Function (RBF) Network: 2 Layers Only



Radial Basis Function (RBF) Network

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



Determination of Parameters

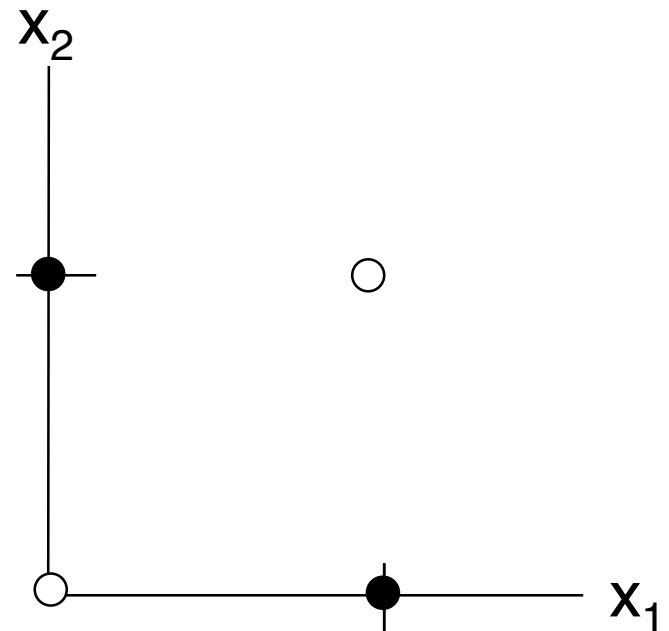
- Given a set of data, the weights, centers, and spreads need to be determined for the best fit.
- Approaches:
 - Solving for all parameters
 - Determining centers and spreads by clustering, then training weights
 - Training for centers, spreads, and weights

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**.

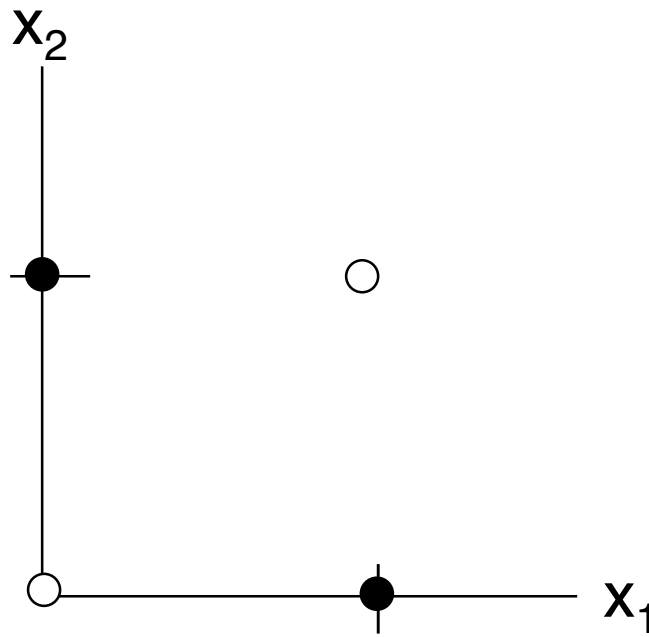
Example: xor

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

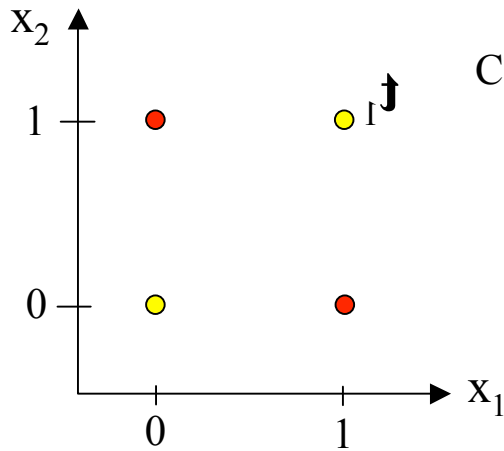


Example: xor

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



XOR with RBF



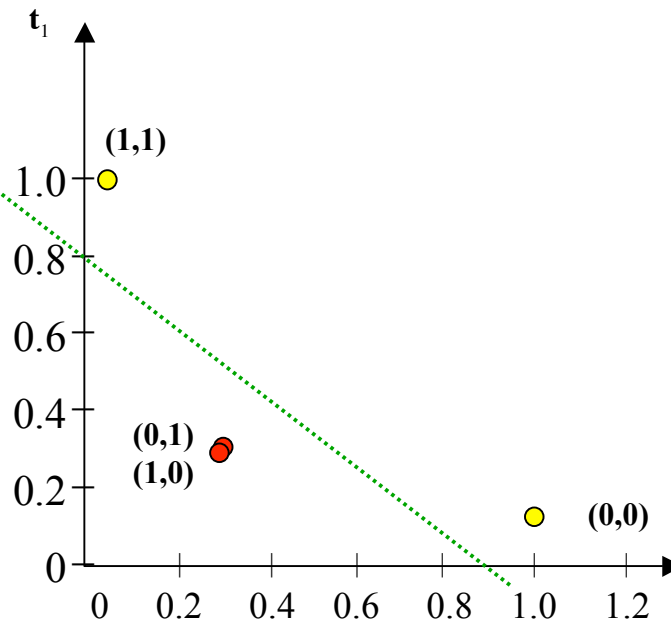
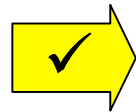
Consider the nonlinear functions to map the input vector \mathbf{x} to the φ_1 - φ_2 space

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

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

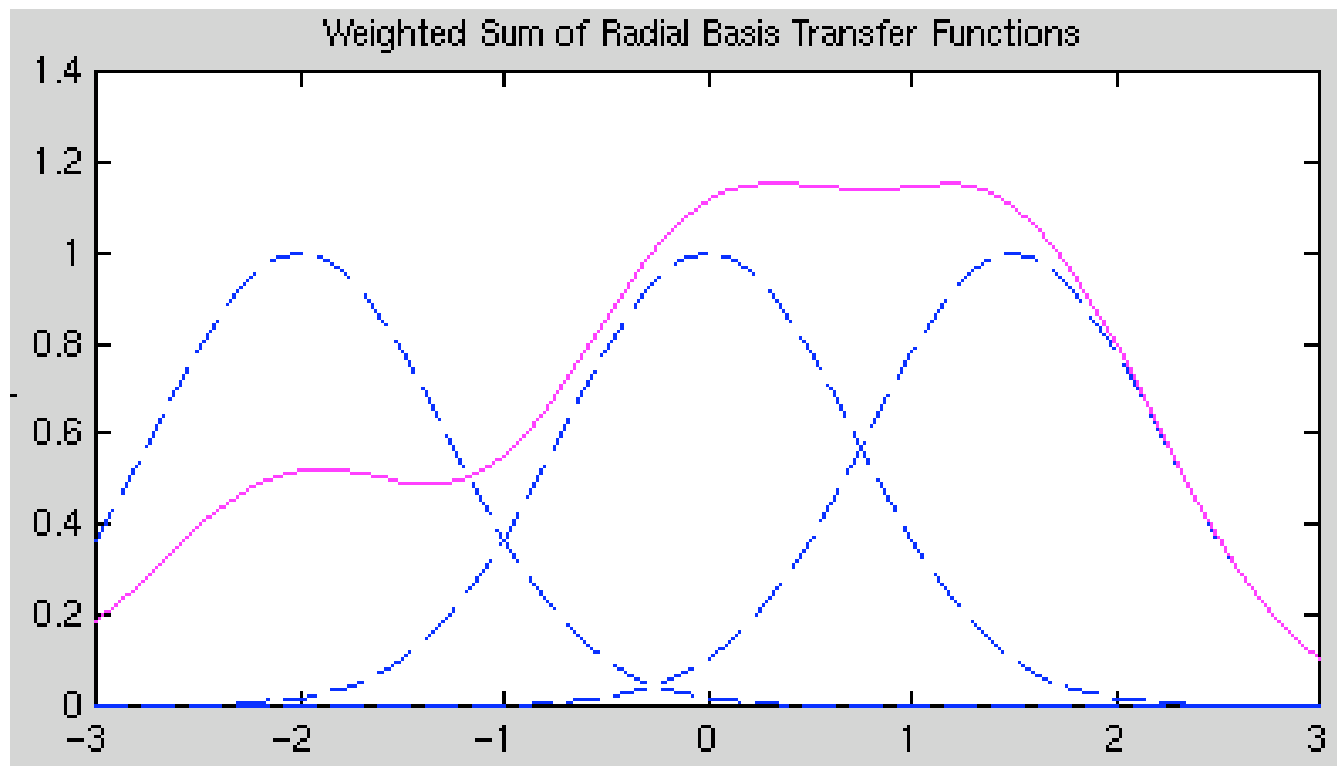
$$\varphi_2(\mathbf{x}) = e^{-\|\mathbf{x} - \mathbf{t}_2\|^2} \quad \mathbf{t}_2 = [0 \ 0]^T$$

Input \mathbf{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

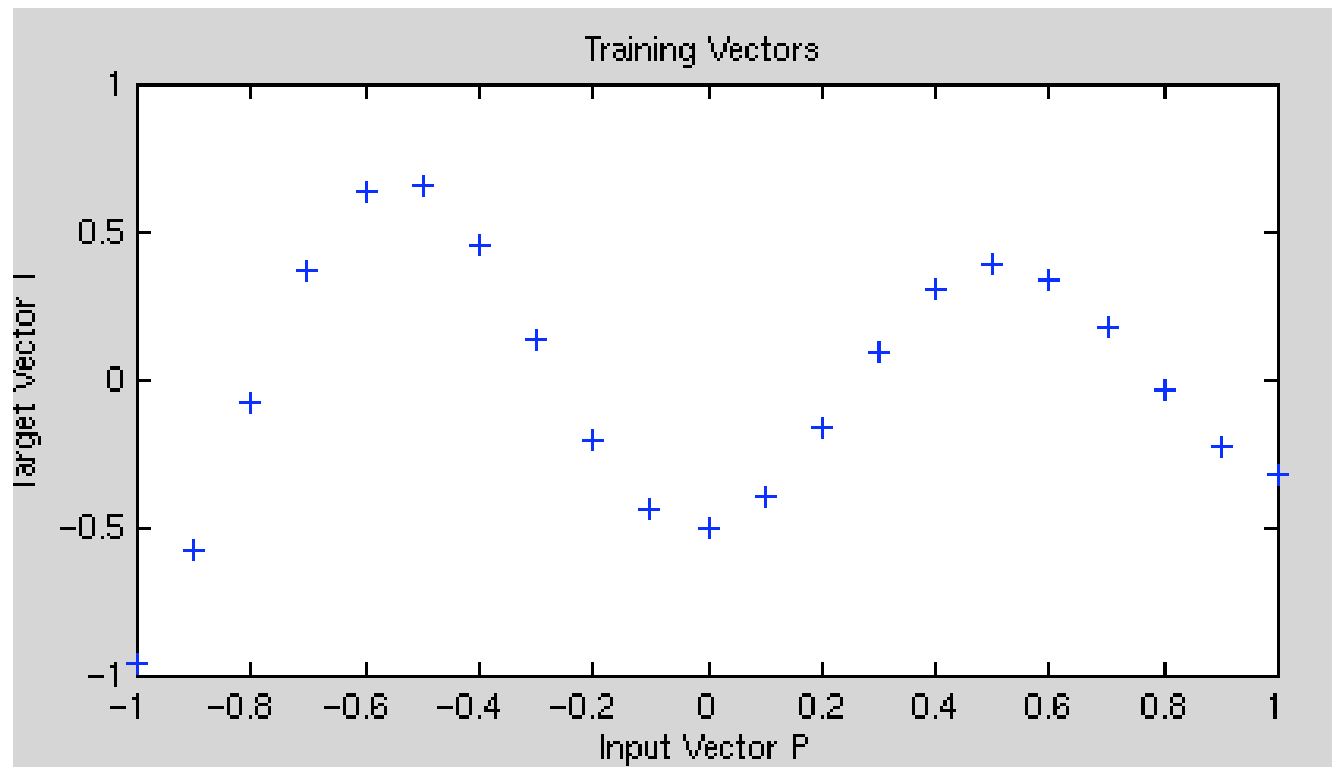


The nonlinear φ function transformed a nonlinearly separable problem into a linearly separable one.

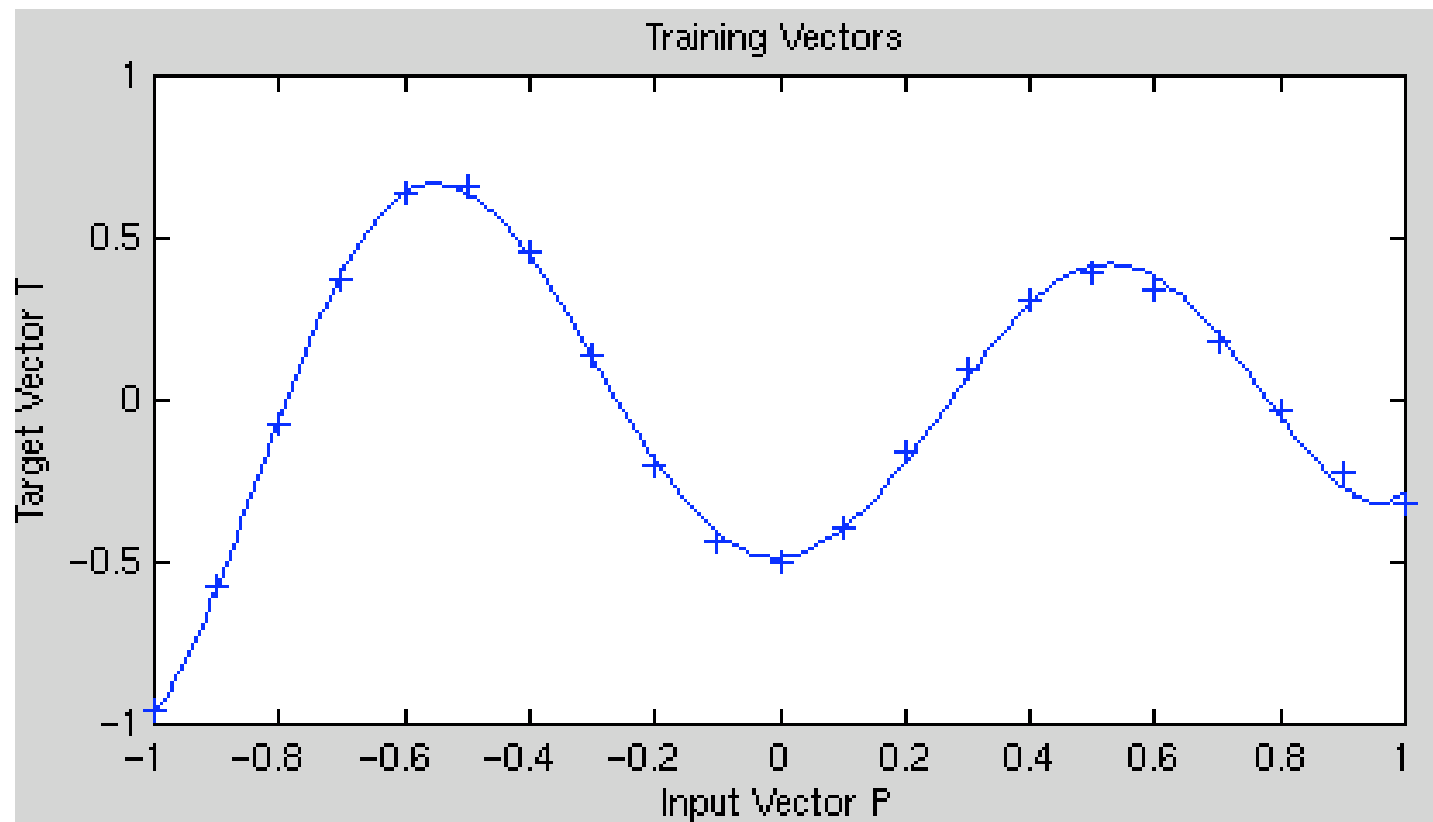
Example: Function Approx.



demorb1



demorb1



RBF Properties

- RBF networks tend to have good **interpolation** properties,
- but not as good **extrapolation** properties as MLP's.

- For extrapolation, using a given number of neurons, an MLP could be a much better fit.

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).

Possible Applications

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

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 RADBAS 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 weights equal to that vector.
- 4) The PURELIN layer weights are redesigned by solving a set of linear equations.

Training Approach for RBFNs

- Haykin, section 5.13, gives update formulas for simultaneously training weights, centers, and spreads iteratively using **gradient descent**.
- The end result is reproduced on the next slides.

Training for Weights (Haykin)

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

$$e_j = d_j - \sum_{i=1}^M w_k \varphi(\|\mathbf{x}_j - \mathbf{t}_i\|)$$

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

(Haykin)

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

1. *Linear weights* (output layer)

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

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

2. *Positions of centers* (hidden layer)

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

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

3. *Spreads of centers* (hidden layer)

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

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

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

Some Wisdom on Training RBFN's

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

A Solving Approach for RBF

- Assume the spreads are fixed.
- Choose the N data points themselves as centers.
- It 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

- \mathbf{w} is the weight vector
 - \mathbf{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

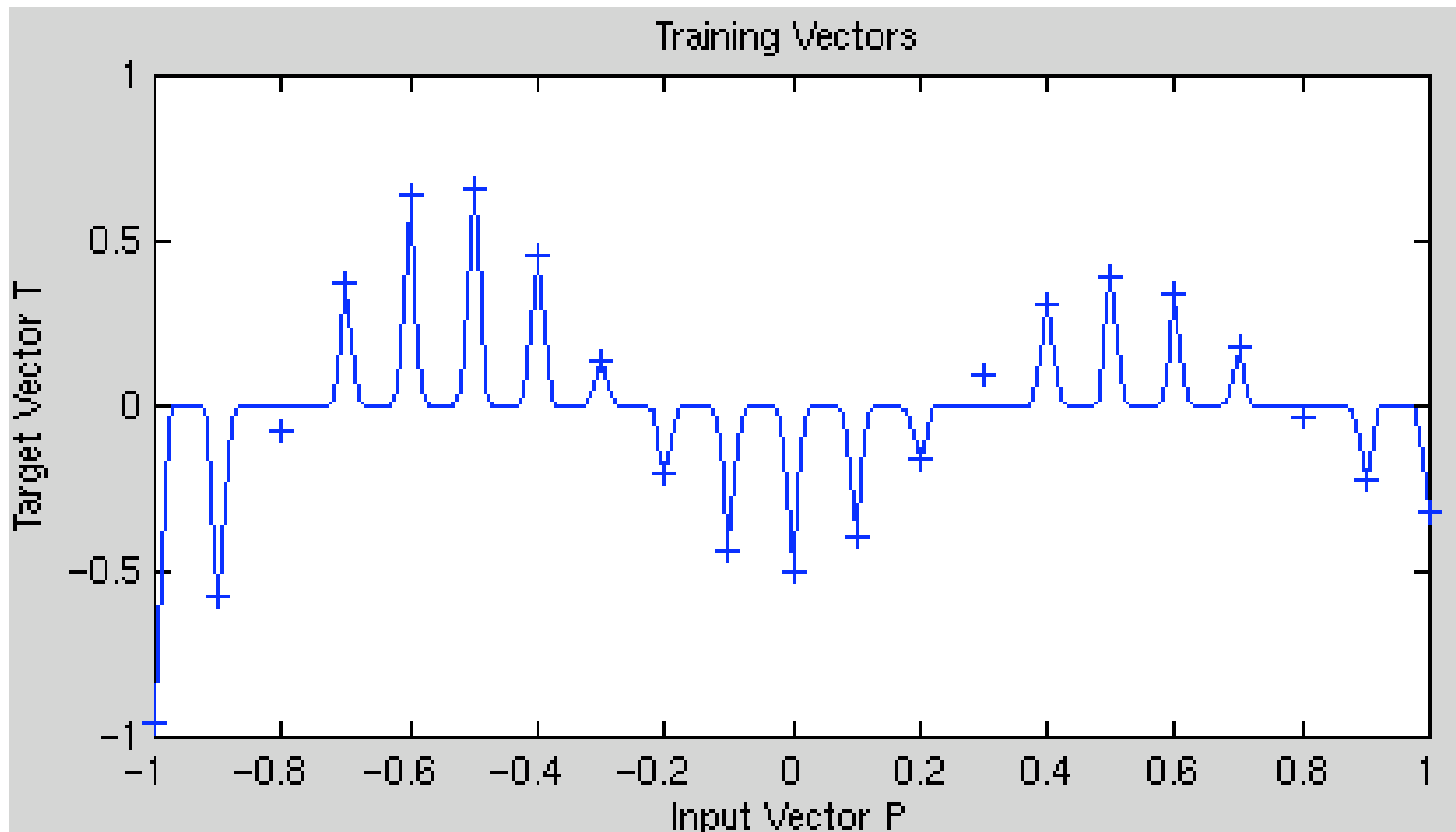
$$\mathbf{w} = \Phi^{-1} \mathbf{d}$$

Solving Approach for RBFNs

- **Micchelli's Theorem** says that if the points x_i are distinct, then the Φ matrix *will* be non-singular.
- Ref: Mhaskar and Micchelli, *Approximation by superposition of sigmoidal and radial basis functions*, Advances in Applied Mathematics, 13, 350-373, 1992.

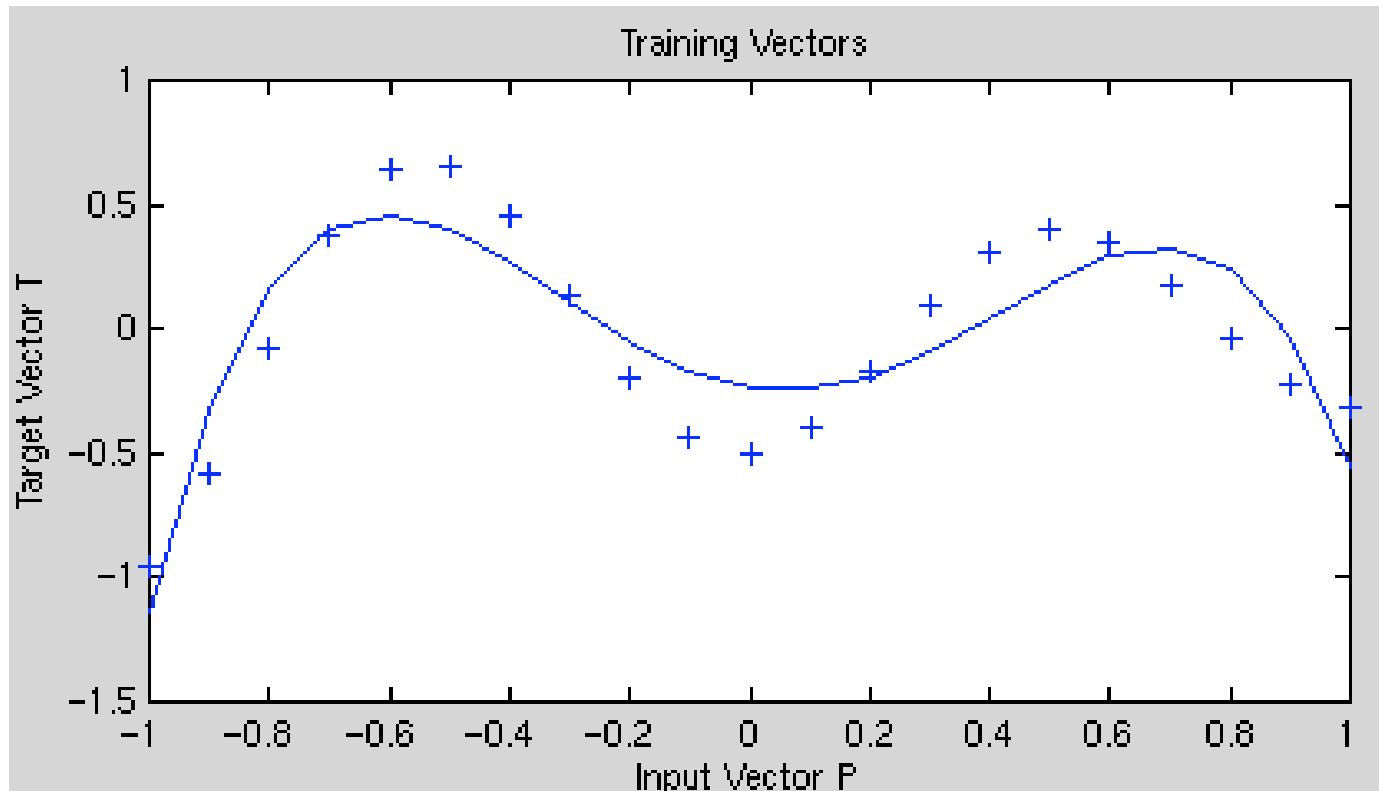
What happens if spreads too small? (demorb3)

Here spreads = 0.01 (vs. 1.0 in previous case). The network does not generalize.



What happens if spreads too large? (demorb4)

Here spreads = 100. The network over-generalizes.



Bias-Variance Dilemma

- “Bias-variance dilemma” is the term used by NAS regarding the choice of spreads.
- (NAS = Neural and Adaptive Systems, Jose C. Principe, Neil R. Euliano, Curt Lefebvre)

Training with 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”.

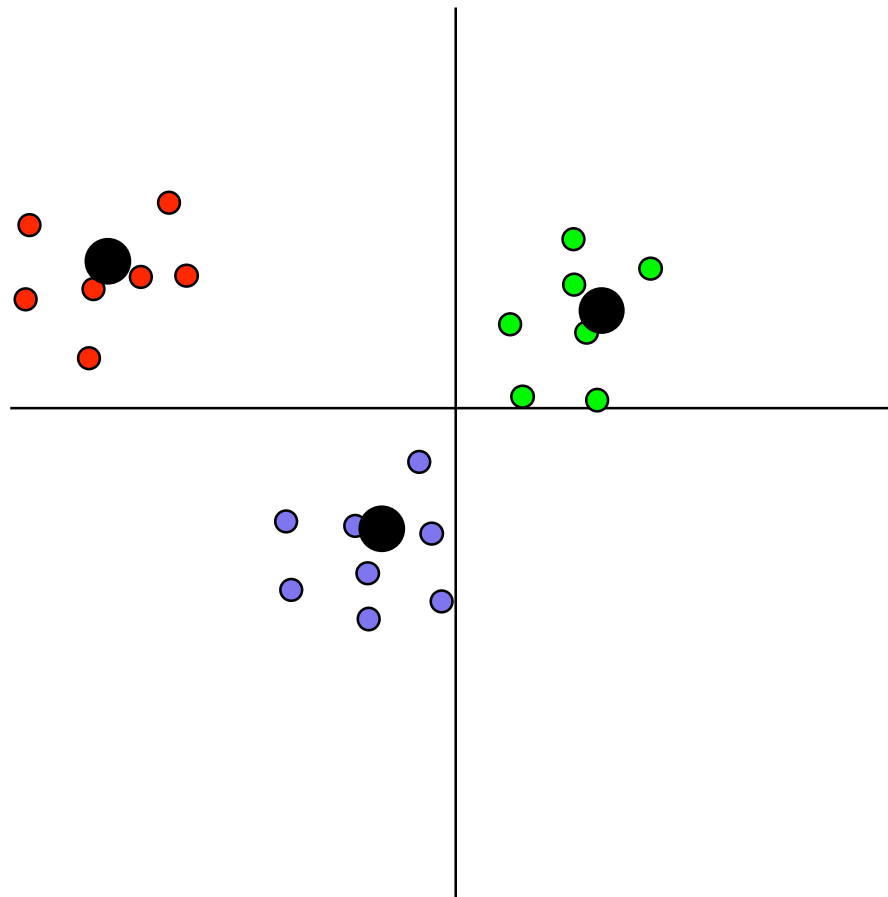
Regularization or Weight Decay

- **NAS 5.12** uses 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 also known as **“regularization”**

Selecting Centers by Clustering

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

Selecting Centers by Clustering



“k-means clustering”

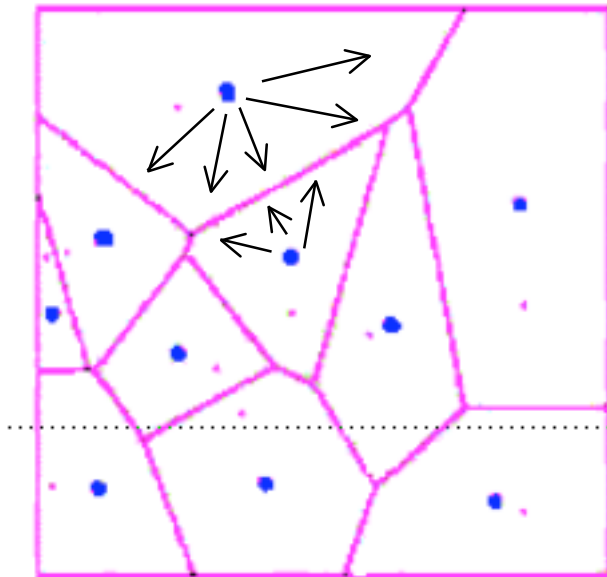
(MacQueen 1967)

- 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.

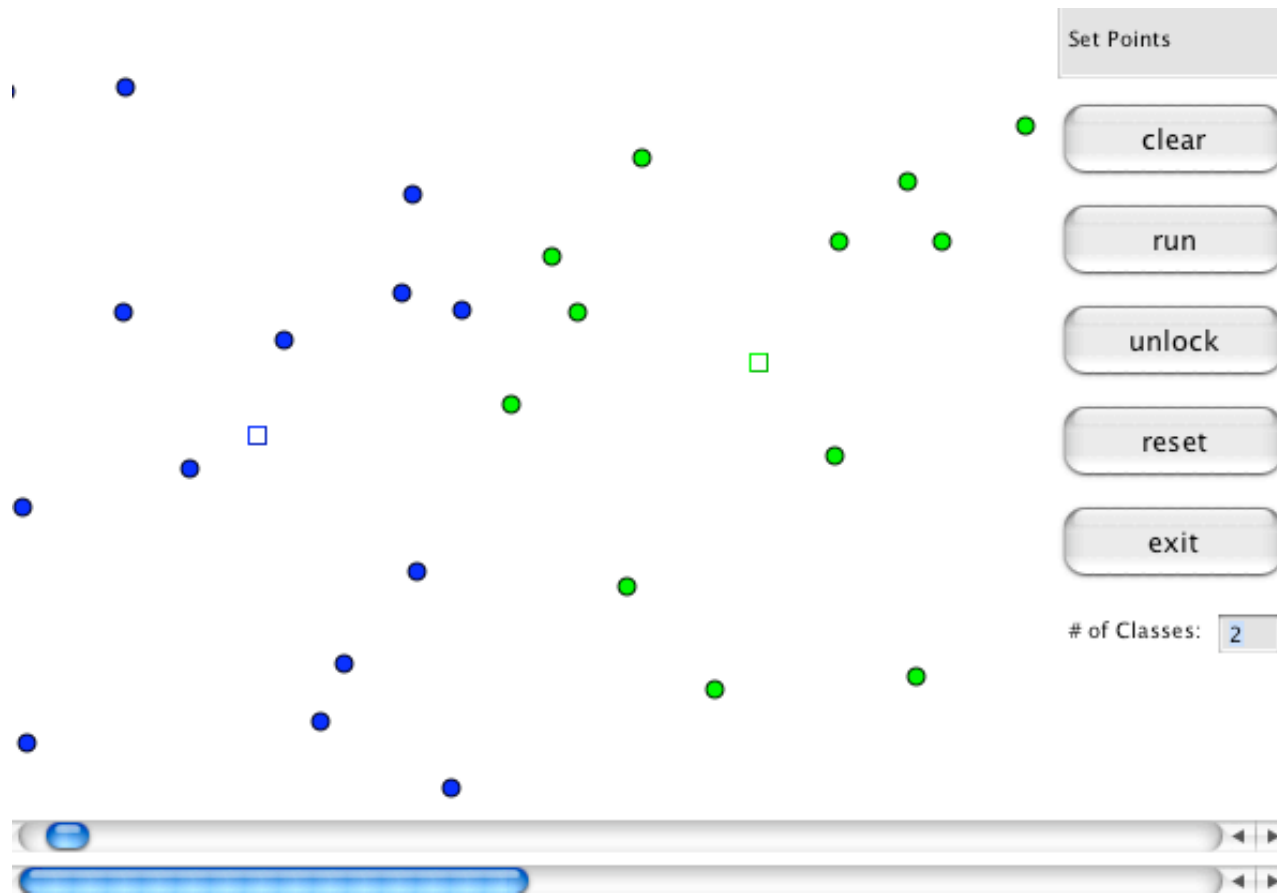
$$E = \sum_{j=1}^K \sum_{i=1}^N (\|v_i - c_j\|)^2$$

- This is a heuristic procedure, and is subject to the usual **local minima** pitfalls.
- However, it is used quite often.

See 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://www.delft-cluster.nl/TextMiner/theory/kmeans/kmeans.html>



Computing weights when more points than centers

- 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 a least squares or “pseudo inverse” technique:

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

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. $c = 5$).

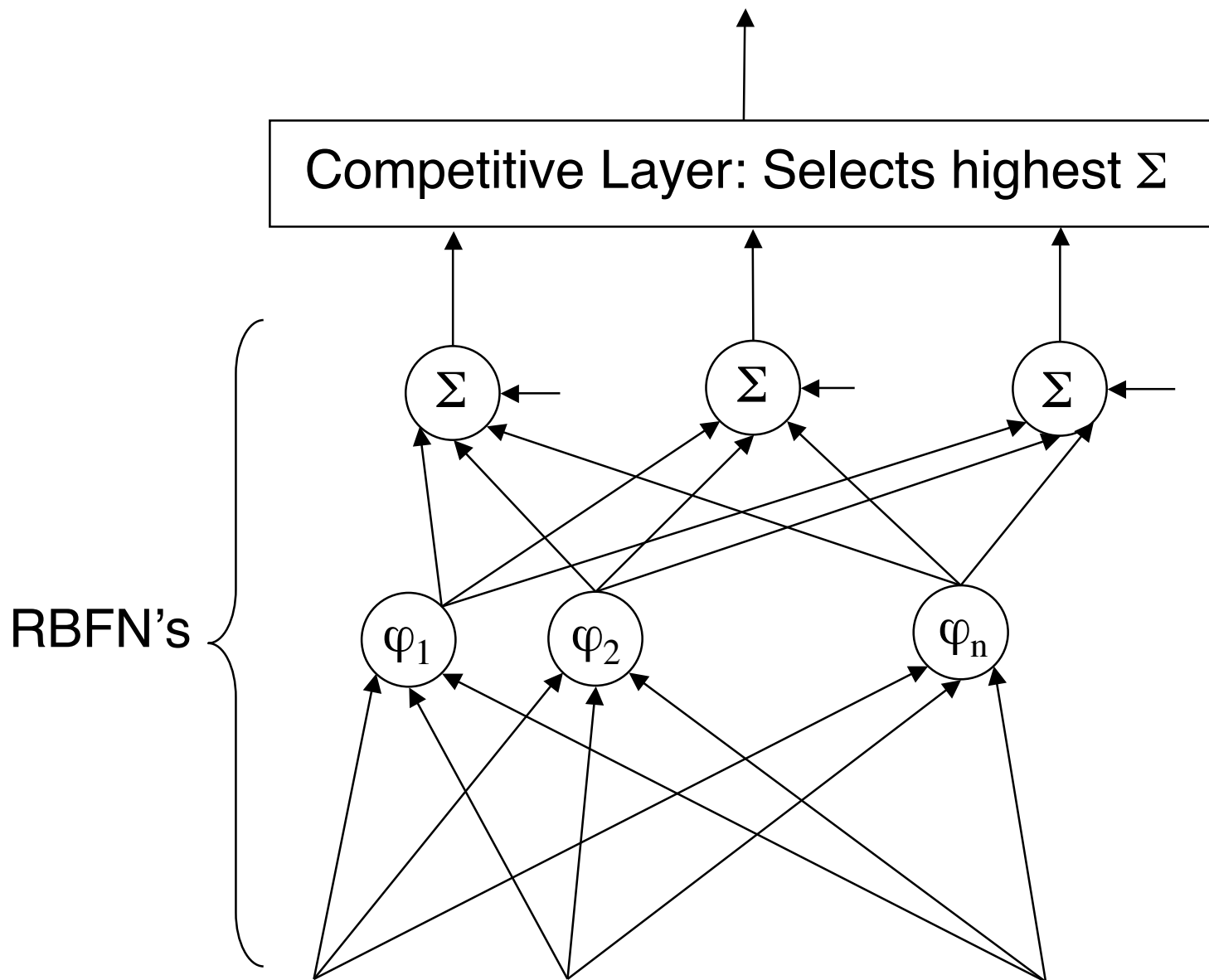
MLP vs RBF Case Studies

(source: Yampolskiy and Novikov, RIT)

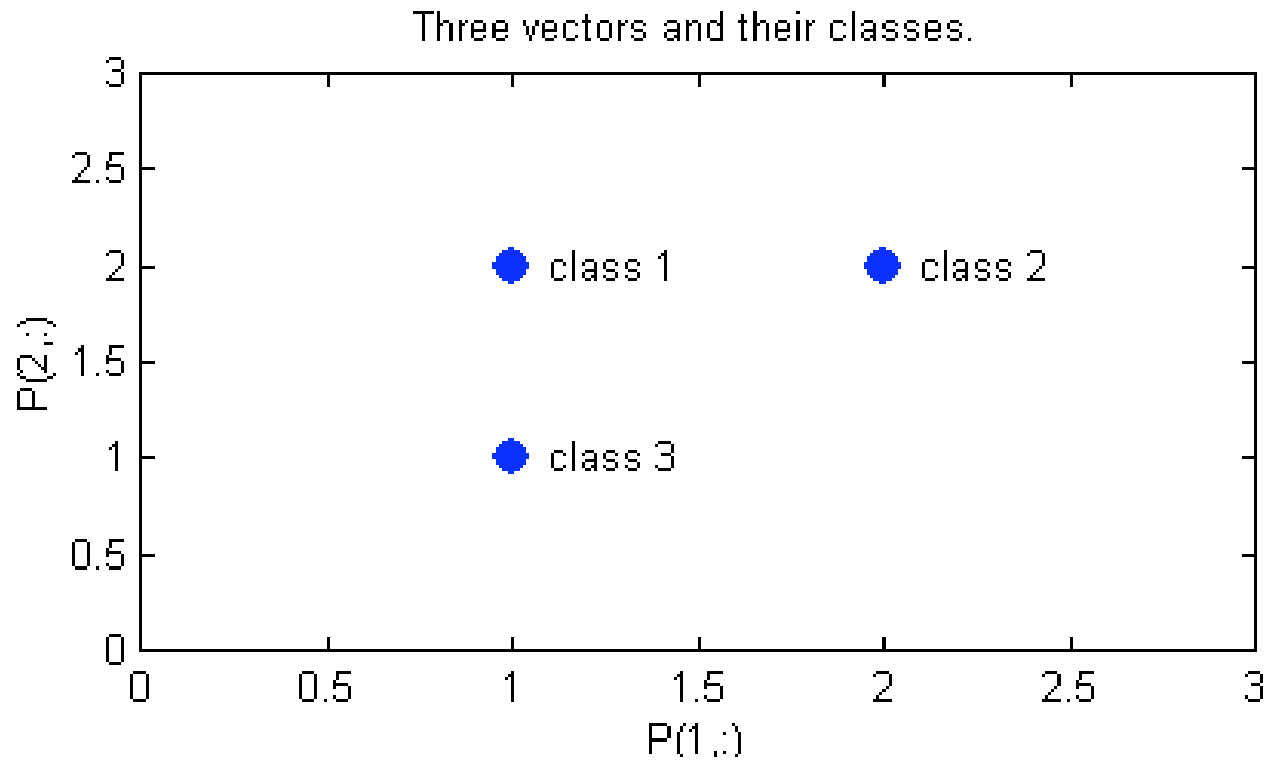
Source	Application	MLP	RBF
Dong	Satellite image classification		Faster runtime
Finan	Speaker recognition		More accurate, less sensitive to bad training data
Hawickhorst	Speech recognition		Faster training, better retention of generalization
Li	Surgical decision making	Fewer hidden nodes	Shorter training 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	

“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**.



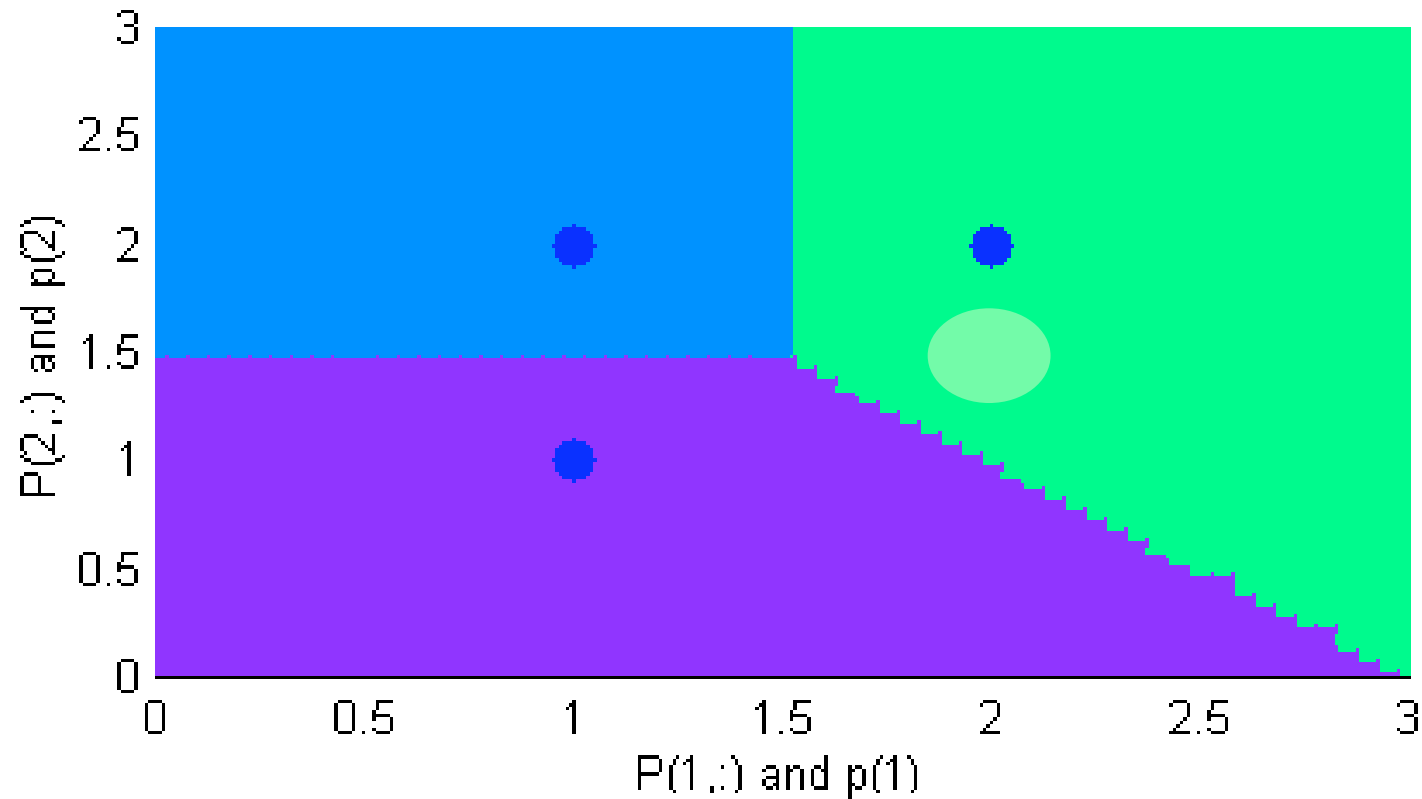
demopnn1



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

demopnn1 classification regions defined by training data

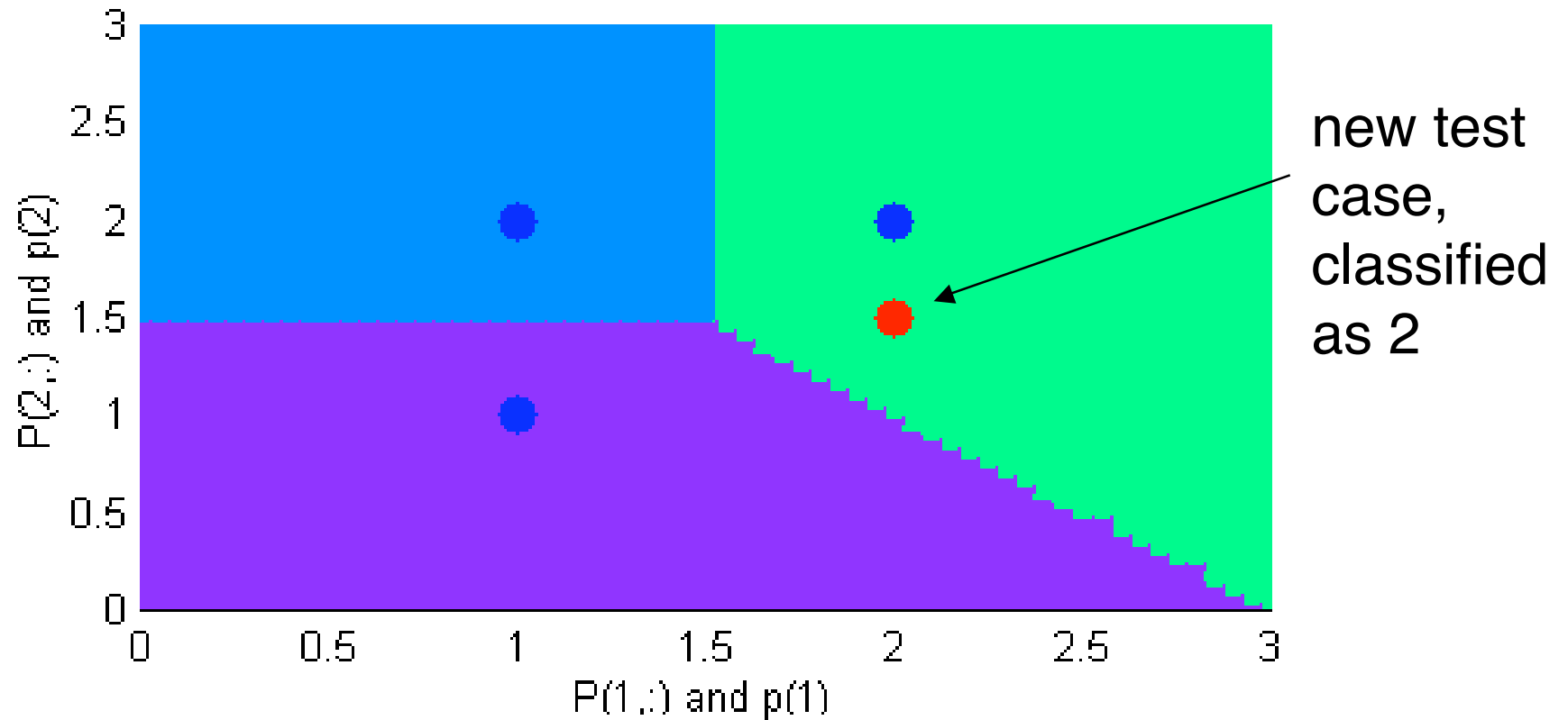
The three classes.



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

demopnn1 on test data

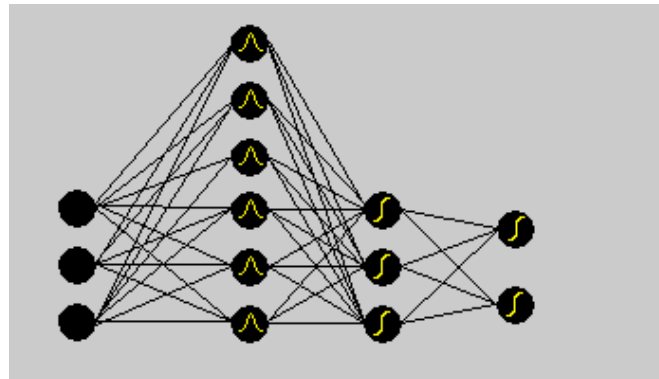
The three classes.



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

GRNN's

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



- They are based statistical estimation theory (Bayesian).
- For matlab, see <http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/radial78.shtml>

Related Topic to GRNN

- Support Vector Machines (SVM)
to be discussed

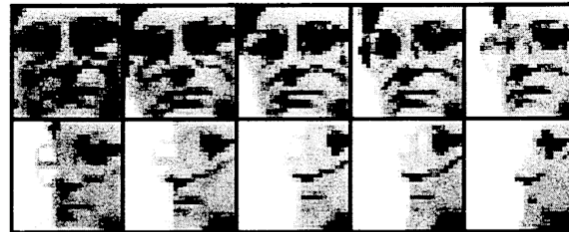
Face Recognition Case Study

(Powell, et al. at University of Sussex)

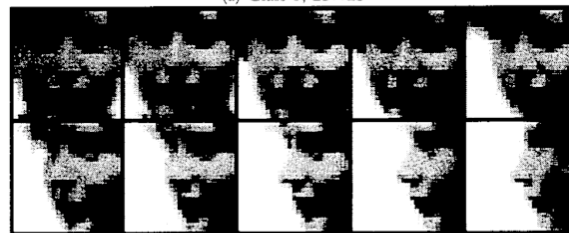
- **Database**

- **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 assess performance of **face recognition techniques** when pose variations occur

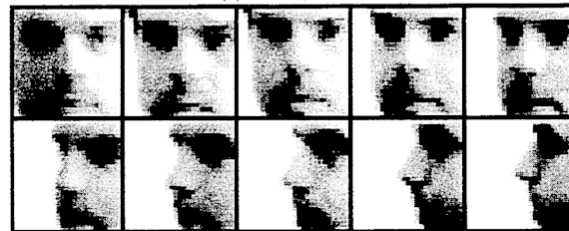
Sample Images (different angles)



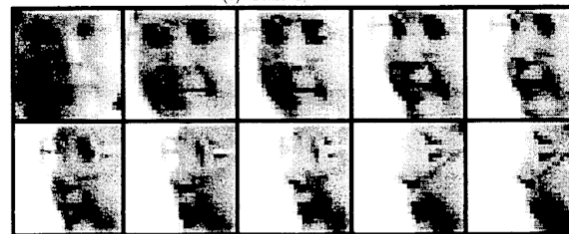
(a) Class 0, 25x25



(b) Class 1, 25x25



(c) Class 2, 25x25



(d) Class 3, 25x25

Approach: Face-unit subnets

- Each face recognition unit RBF neural 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 inputs which represent the pixel intensities (normalized) of an image.
- Hidden layer contains $p+a$ neurons:
 - p hidden pro-neurons (receptors for positive evidence)
 - a hidden 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, discarded 23 percent of the images of the test set and classified correctly 96 percent of the non discarded images.