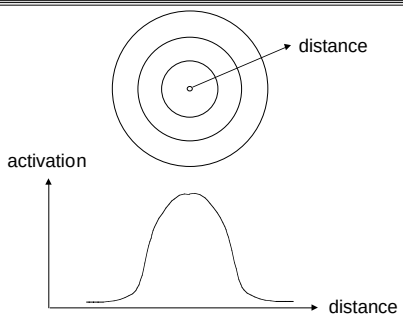
# Radial Basis Function Networks

## Radial Basis Functions

- Contrast to sigmoids, 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.
- Models on-center off-surround phenomenon found in certain real neurons in the visual system

## On-Center, Off-Surround

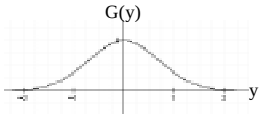



## Modeling

- $\varphi_i(x) = G( \| x - c_i \| )$

  where G is a decreasing function and $c_i$ is the **center**.
- Example: Gaussian:

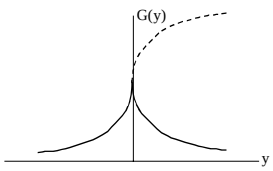
  $G(y) = \exp(-y^2/\sigma^2)$




  where $\sigma$ is a parameter called the **spread**, which indicates the **selectivity** of the neuron.

## Other RBF Examples

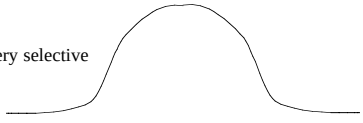
- $G(y) = 1/\text{sqrt}(y^2 + \sigma^2)$
- $G(y) = 1/(1+\exp(ay^2))$ "reflected sigmoid"



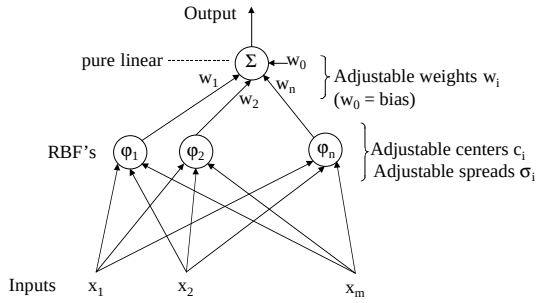

## Spread = 1/Selectivity

Small Spread, very selective

Large Spread, not very selective

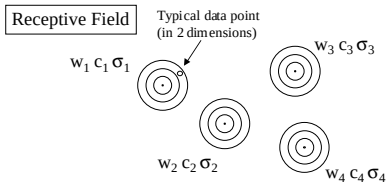


## Radial Basis Function (RBF) Network: 2 Layers Only

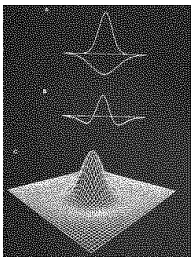



## Radial Basis Function (RBF) Network

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

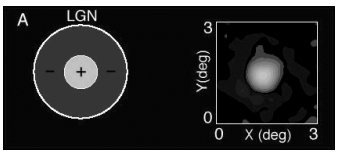



## 3-D depiction of 2-D on-center



## On-Center response captured in a lab experiment

(see 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.

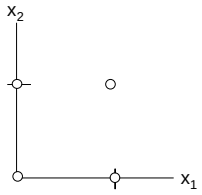
The receptive fields of the LGN neurons are **circularly symmetric** and have the same **center-surround** organization. The algebraic sum of the center and surround mechanisms has a vague resemblance to a **sombrero** with a tall peak, so this model of the receptive field is sometimes called "**Mexican-hat 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.

## On-Center Behavior

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

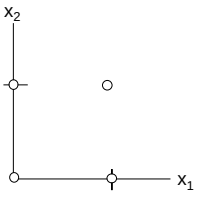
## Example: xor

- 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.

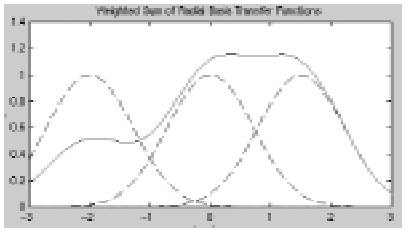



## Example: xor

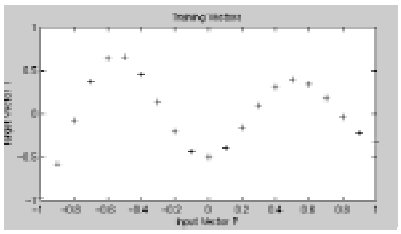
- Choose centers at (1, 0), and (1, 0). Choose spreads as, say 0.1, find weights.



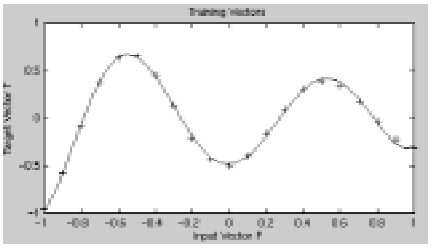

## Example: Function Approx.



## demorb1



## demorb1



## RBF Properties

- RBF networks tend to have good **interpolation** properties,
- but not as good **extrapolation** properties.
- For extrapolation, using a given number of neurons, an MLP could be much better.

## 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.

## Possible Applications

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

## 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
  - Determining centers and spreads, 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, there may be one node per training element.

## Example: matlab newrb

NEWRB Design a radial basis network.

net = newrb(P,T,GOAL,SPREAD)

**NEWRB adds neurons to the hidden layer of a radial basis network until it meets the specified mean squared error goal.**

NEWRB(PR,T,GOAL,SPREAD) takes two to four 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.
and returns a new radial basis network.

## 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.

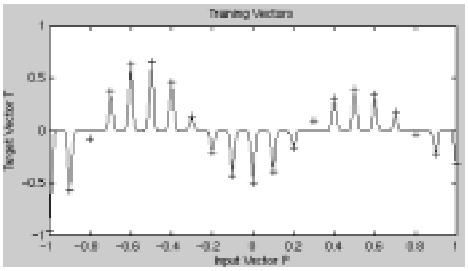
2) The input vector with the greatest error is found.

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

4) The PURELIN layer weights are redesigned to minimize error.

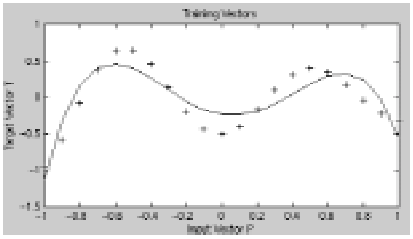
## What happens if spreads too small? (demorb3)

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




## What happens if spreads too large?

Here spreads = 100. The network does not train well.




## Bias-Variance Dilemma

- NAS 5.6.1 calls the tension the “bias-variance dilemma.

## Matlab advice

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.

## More Examples

- See NAS, section 5.3.4.
- NAS Examples 5.6, 5.9, 5.12, 5.13

## 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”.

## Weight Decay

- **NAS 5.12** uses the Weight Decay method (NAS Ex. 4.16) to prune an RBF:
  - At each update, a small amount is subtracted 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**"

## 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 $\varphi$ is the radial basis function.
- The matrix $\Phi$ of values $\varphi_{ji}$ is called the **interpolation matrix**.

## Solving Approach for RBF

- The matrix $\Phi$ of values $\varphi_{ji}$ is called the **interpolation matrix**. It has the property that

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

  where **w** is the weight vector and **d** is the desired output.
- If $\Phi$ is non-singular, then we have our weights as

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

## Solving Approach for RBFNs

- A theorem known as **Micchelli's Theorem** says that if the points $x_i$ are distinct, then the $\Phi$ 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.

## Training Approach for RBFNs

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

**TABLE 5.4** Adaptation Formulas for the Linear Weights and the Positions and Spreads of Centers for RBF Network[a]

1. *Linear weights* (output layer)

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

$$w_i(n+1) = w_i(n) - \eta_1 \frac{\partial \mathcal{E}(n)}{\partial w_i(n)}, \qquad i = 1, 2, \ldots, \mathbf{m}_1$$

2. *Positions of centers* (hidden layer)

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

$$\mathbf{t}_i(n+1) = \mathbf{t}_i(n) - \eta_2 \frac{\partial \mathcal{E}(n)}{\partial \mathbf{t}_i(n)}, \qquad i = 1, 2, \ldots, \mathbf{m}_1$$

3. *Spreads of centers* (hidden layer)

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

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

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

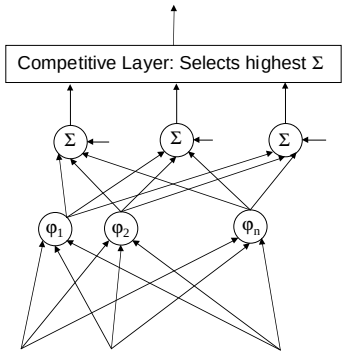
[a]The term $e_j(n)$ is the error signal of output unit $j$ at time $n$. The term $G'(\cdot)$ is the first derivative of the Green's function $G(\cdot)$ with respect to its argument.

## Modified Approach

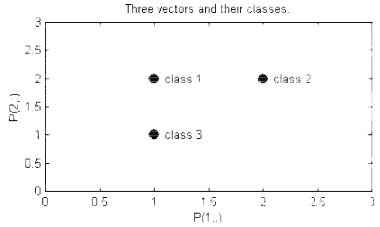
- An approach that has also been successful is to **first establish the centers** by another method, then train the weights.
- Training centers can be done by "Competitive learning" which we study next.

## Probabilistic Neural Networks

- A Probabilistic Neural Network (PNN) is simply 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.

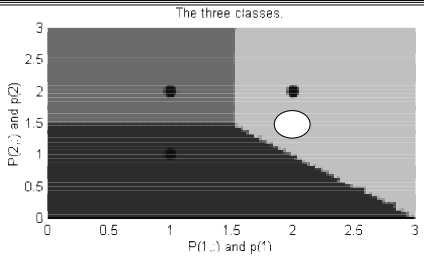



## demopnn1



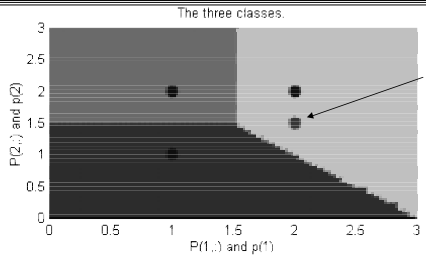

```
P = [1 2; 2 2; 1 1]'; T = ind2vec([1 2 3]);
net = newpnn(P,T,spread);
```

## demopnn1 on training data




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

## demopnn1 on test data




```
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).
- More on this topic later.