Radial Basis Function Neural Networks

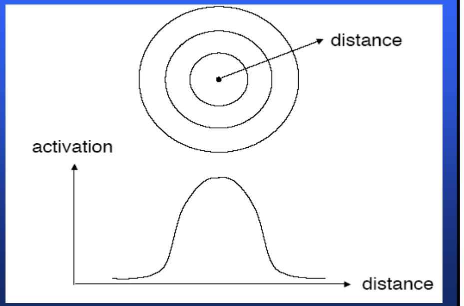
Topic 8-3

Note: lecture notes by Michael Negnevitsky (U of Tasmania, Australia) and Bob Keller (Harvey Mudd College, CA) are used

Main idea: change the activation function

- 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.

Main idea: geometry



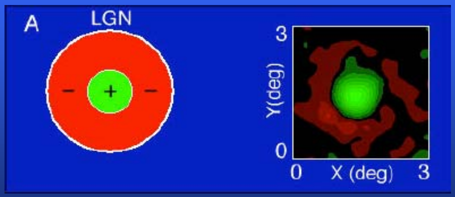
On-Center response in a lab

LGN (lateral geniculate nucleus) 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.

LGN response



Modeling

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

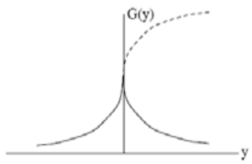
  where G is a decreasing function and $\mathbf{c}_i$ is the **center**.
- Example: Gaussian:

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

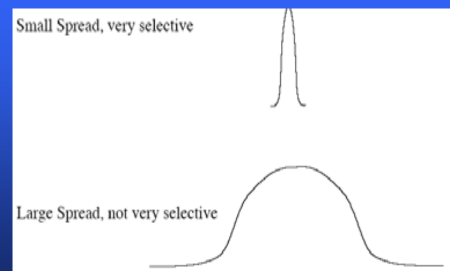
where σ is a parameter called the **spread**, which indicates the **selectivity** of the neuron.

Other RBF examples

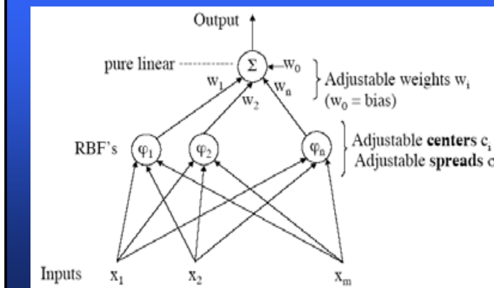
- $G(y) = 1/\mathrm{sqrt}(y^2 + \sigma^2)$
- $G(y) = 1/(1+\exp(ay^2))$ "reflected sigmoid"



Spread = 1/selectivity

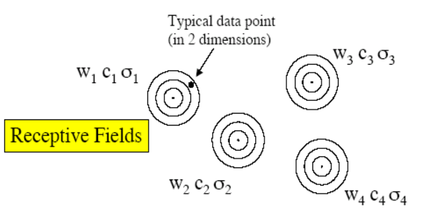


RBF Network : two layers only



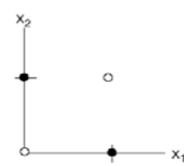
RBF network

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



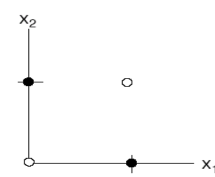
Example: XOR with RBF

- 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.

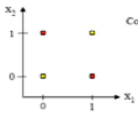


Example: XOR with RBF

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



Example: XOR with RBF

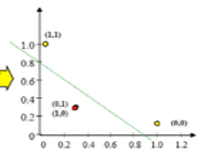


Consider the nonlinear functions to map the input vector $\mathbf{x}$ to the φ_1- φ_2 space

$$\mathbf{x}=[x_1\ x_2] \quad \varphi_1(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{t}_1\|^2} \quad \mathbf{t}_1 = [1\ 1]^T$$

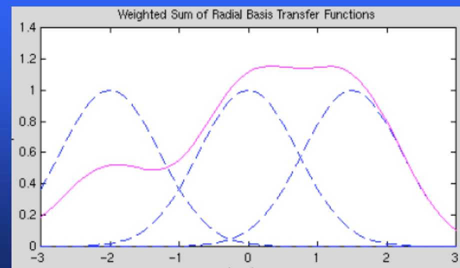
$$\varphi_2(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{t}_2\|^2} \quad \mathbf{t}_2 = [0\ 0]^T$$

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

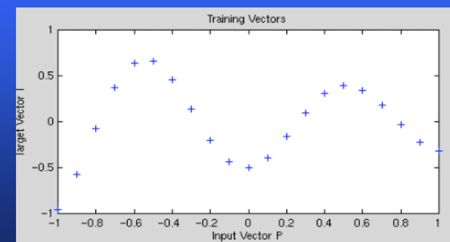


The nonlinear φ function transformed a nonlinearly separable problem into a linearly separable one !!!

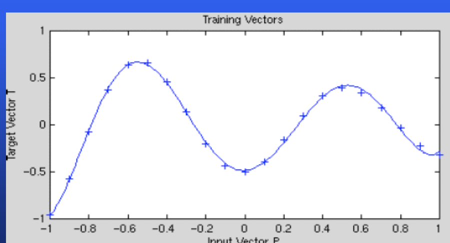
Example: Function approximation



demo



demo



RBF properties

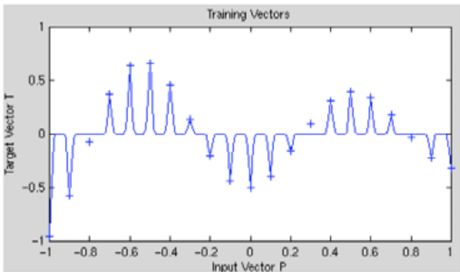
- RBF networks tend to have good **interpolation** properties, but not as good **extrapolation** properties as MLP’s. For extrapolation, using a given number of neurons, an MLP could be a much better fit.
- With proper setup, RBFNs can train in time **orders of magnitude faster** than backpropagation.
- RBFNs enjoy the same **universal approximation** properties as MLPs: given sufficient neurons, any reasonable function can be approximated (with just 2 layers).

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.
```

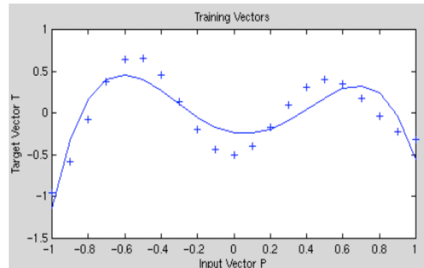
Demo: spreads are too small

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



Demo: spreads are too large

Here spreads = 100. The network over-generalizes.



RBF training for weights, centers and spreads using gradient descent

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

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

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

G' represents the first derivative of the function wrt its argument

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

1. *Linear weights* (output layer)

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

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

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)

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

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

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

Some tricks on RBF NN training

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

Solving approach for RBF NN

- 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 NN

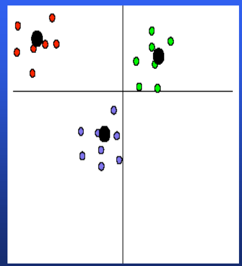
- The **interpolation matrix** has the property that
  $\Phi \mathbf{w} = \mathbf{d}$ where
  w is the weight vector
  d is the desired output vector over all training samples (since the samples are both data points and centers).
- If Φ is non-singular, then we can **solve** for weights as $\mathbf{w} = \Phi^{-1}\, \mathbf{d}$

Bias-Variance dilemma or how to choose the numbers

- Two devils: approximation error vs. overfitting on training set
- Reason for overfitting: too large model does not get an ability to generalize
- How to discover this: while moving from training to testing set
- Errors do increase but should not too much

Selecting centers by clustering

- One center per training sample may be overkill.
- There are ways to select centers as representatives among **clusters**, given say a fixed number of representatives.



K-means clustering

- 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.

K-means clustering

The Voronoi Tesselation 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**.

Tries to optimize the SSE of the difference between points and the center of their cluster

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

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

However, it is used quite often.

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 retraing time
Roppel	Odor recognition	Higher identification rates	