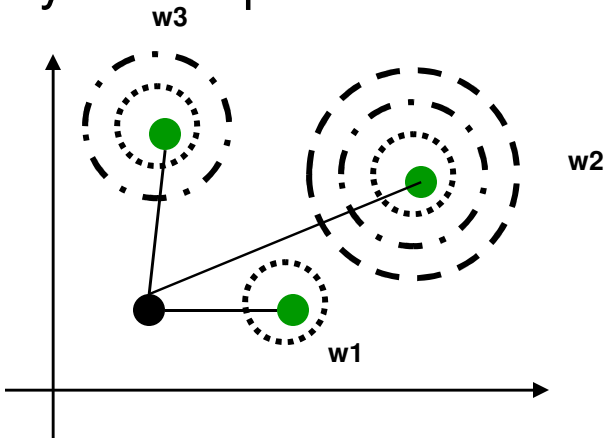
Radial Basis Function

Radial-Basis Function Networks

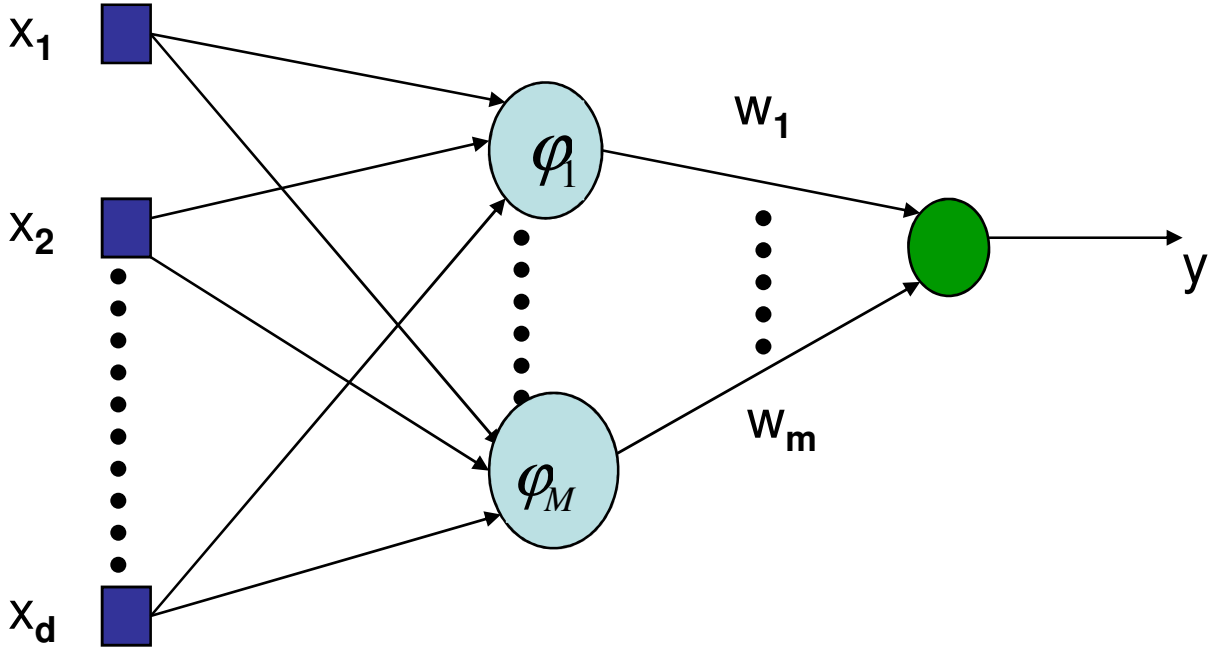
- A function is radial basis (RBF) if its output depends on (is a non-increasing function of) the distance of the input from a given stored vector.
- RBFs represent local receptors, as illustrated below, where each green point is a stored vector used in one RBF.
- In a RBF network one hidden layer uses neurons with RBF activation functions describing local receptors. Then one output node is used to combine linearly the outputs of the hidden neurons.



The output of the red vector is "interpolated" using the three green vectors, where each vector gives a contribution that depends on its weight and on its distance from the red point. In the picture we have

$$w1 < w3 < w2$$

RBF ARCHITECTURE



- One hidden layer with RBF activation functions $\varphi_1 \ldots \varphi_M$
- Output layer with linear activation function.

$$y = w_1 \varphi_1 (\| \mathbf{x} - \mathbf{t}_1 \|) + \ldots + w_M \varphi_M (\| \mathbf{x} - \mathbf{t}_M \|)$$

$$\| \mathbf{x} - \mathbf{t} \| \text{ distance of } \mathbf{x} = (x_1, \ldots, x_d) \text{ from vector } \mathbf{t}$$

Types of φ

- Multiquadrics:

$$\varphi(r) = (r^2 + c^2)^{1/2} \qquad c > 0$$

$$r = \| \mathbf{x} - \mathbf{t} \|$$

- Inverse multiquadrics:

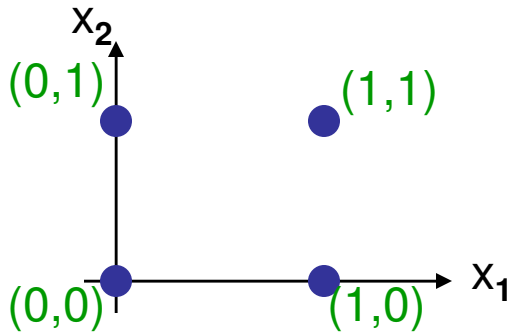
$$\varphi(r) = \frac{1}{(r^2 + c^2)^{1/2}} \qquad c > 0$$

- Gaussian functions (most used):

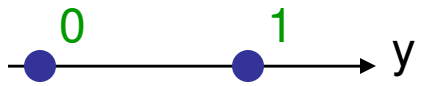
$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \qquad \sigma > 0$$

Example: the XOR problem

- Input space:



- Output space:



- Construct an RBF pattern classifier such that:

  (0,0) and (1,1) are mapped to 0, class C1

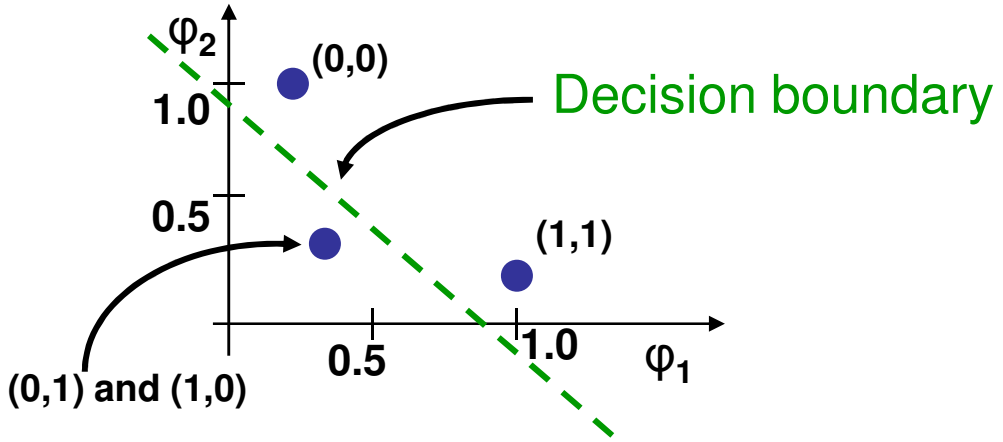
  (1,0) and (0,1) are mapped to 1, class C2

Example: the XOR problem

- In the feature (hidden layer) space:

$$\varphi_1(\|\mathbf{x}-\mathbf{t}_1\|) = e^{-\|\mathbf{x}-\mathbf{t}_1\|^2}$$

$$\varphi_2(\|\mathbf{x}-\mathbf{t}_2\|) = e^{-\|\mathbf{x}-\mathbf{t}_2\|^2} \qquad \text{with } \mathbf{t}_1 = (1,1) \text{ and } \mathbf{t}_2 = (0,0)$$

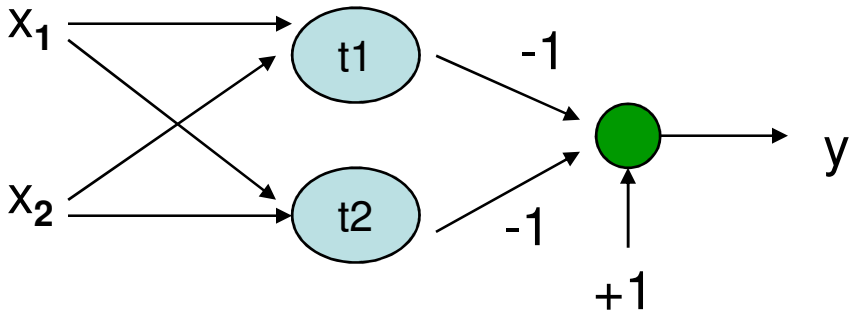


- When mapped into the feature space < φ_1 , φ_2 > (hidden layer), C1 and C2 become *linearly separable.* So a linear classifier with $\varphi_1(x)$ and $\varphi_2(x)$ as inputs can be used to solve the XOR problem.

RBF NN for the XOR problem

$$\varphi_1(\|\mathbf{x}-\mathbf{t}_1\|) = e^{-\|\mathbf{x}-\mathbf{t}_1\|^2} \qquad \text{with } \mathbf{t}_1 = (1,1) \text{ and } \mathbf{t}_2 = (0,0)$$

$$\varphi_2(\|\mathbf{x}-\mathbf{t}_2\|) = e^{-\|\mathbf{x}-\mathbf{t}_2\|^2}$$



$$y = -e^{-\|\mathbf{x}-\mathbf{t}_1\|^2} - e^{-\|\mathbf{x}-\mathbf{t}_2\|^2} + 1$$

If $y > 0$ then class 1 otherwise class 0

RBF network parameters

- **What do we have to learn for a RBF NN with a given architecture?**
  - The centers of the RBF activation functions
  - the spreads of the Gaussian RBF activation functions
  - the weights from the hidden to the output layer
- Different learning algorithms may be used for learning the RBF network parameters. We describe three possible methods for learning centers, spreads and weights.

Learning centre and spread of Gaussian RBFs

The centers of the RBF activation functions and the spreads of the Gaussian RBF activation functions are learnt using a k means algorithm.

Learning weights

- Weights: are computed by means of the pseudo-inverse method.
  - For an example $(\mathbf{x}_i, d_i)$ consider the output of the network

$$y(\mathbf{x}_i) = w_1\varphi_1(\| \mathbf{x}_i - \mathbf{t}_1 \|) + \ldots + w_{m1}\varphi_{m1}(\| \mathbf{x}_i - \mathbf{t}_M \|)$$

  - We would like $y(x_i) = d_i$ for each example, that is

$$w_1\varphi_1(\| \mathbf{x}_i - \mathbf{t}_1 \|) + \ldots + w_M\varphi_M(\| \mathbf{x}_i - \mathbf{t}_M \|) = d_i$$

Learning weights

- This can be re-written in matrix form for one example

and

$$\left[\varphi_1(\| \mathbf{x}_i - \mathbf{t}_1 \|) \ldots \varphi_{m1}(\| \mathbf{x}_i - \mathbf{t}_M \|)\right] \begin{bmatrix} w_1 \\ w_2 \\ \\ w_M \end{bmatrix} = d_i$$

for all the examples at the same time

$$\begin{bmatrix} \varphi_1(\|\mathbf{x}_1-\mathbf{t}_1\|).....\varphi_M(\|\mathbf{x}_1-\mathbf{t}_M\|) \\ ... \\ \varphi_1(\|\mathbf{x}_N-\mathbf{t}_1\|).....\varphi_M(\|\mathbf{x}_N-\mathbf{t}_M\|) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \\ \\ w_M \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \\ \\ d_N \end{bmatrix}$$

Solve using pseudo inverse method, that we did in regression !

Learning by gradient method

- **Apply the gradient descent method for finding centers, spread and weights, by minimizing the (instantaneous) squared error** $E = \frac{1}{2}(y(x) - d)^2$
- **Update for:**

centers

$$\Delta t_j = -\eta_{t_j} \frac{\partial E}{\partial t_j}$$

spread

$$\Delta \sigma_j = -\eta_{\sigma_j} \frac{\partial E}{\partial \sigma_j}$$

weights

$$\Delta w_{ij} = -\eta_{ij} \frac{\partial E}{\partial w_{ij}}$$

Comparison with multilayer NN

- Architecture:
  - RBF networks have one *single* hidden layer.
  - FFNN networks may have *more* hidden layers.

- Neuron Model:
  - In RBF the neuron model of the hidden neurons is *different* from the one of the output nodes.
  - Typically in FFNN hidden and output neurons share a *common neuron model*.
  - The hidden layer of RBF is *non-linear*, the output layer of RBF is *linear*.
  - Hidden and output layers of FFNN are usually *non-linear*.

Comparison with multilayer NN

- Activation functions:
  - The argument of activation function of each hidden neuron in a RBF NN computes the *Euclidean distance* between input vector and the center of that unit.
  - The argument of the activation function of each hidden neuron in a FFNN computes the *inner product* of input vector and the synaptic weight vector of that neuron.

- Approximation:
  - RBF NN using Gaussian functions construct *local* approximations to non-linear I/O mapping.
  - FF NN construct *global* approximations to non-linear I/O mapping.