Department of Computer Engineering
University of Kurdistan

Neural Networks (Graduate level)

Radial Basis Function Networks

By: Dr. Alireza Abdollahpouri

RBF networks

- Radial basis function network (RBFN) represent a special category of the feedforward neural networks architecture.

- The basic RBFN structure consists of an input layer, a <u>single</u> hidden layer with radial activation function and an output layer.

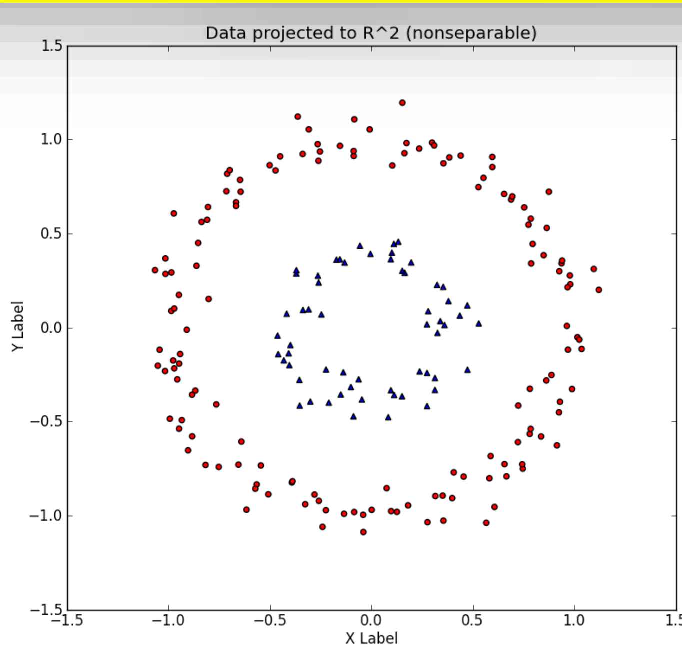
RBF networks

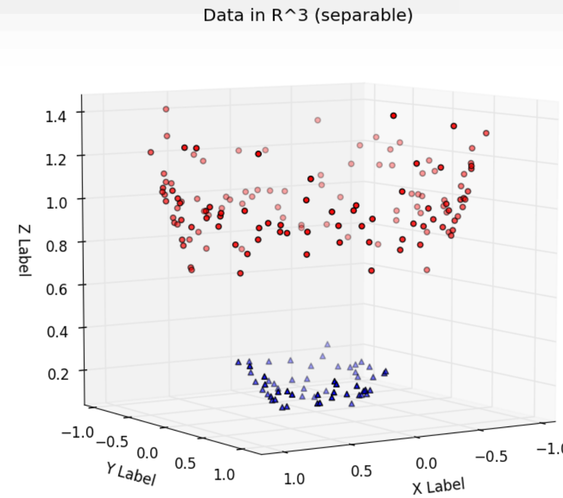
A complex pattern-classification problem cast in a high-dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space

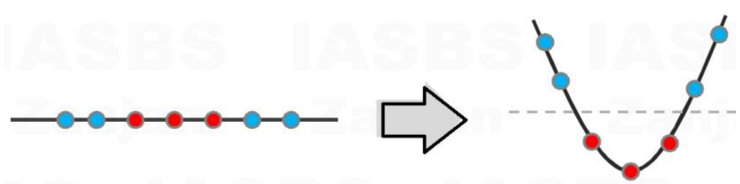
- Implementing this procedure using a network architecture, yields the RBF networks, if the nonlinear mapping functions are **radial basis functions**.
- Radial Basis Functions:
  - **Radial**: Symmetric around its center
  - **Basis Functions**: A set of functions whose linear combination can generate an arbitrary function in a given function space.

RBF networks

A complex pattern-classification problem cast in a high-dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space





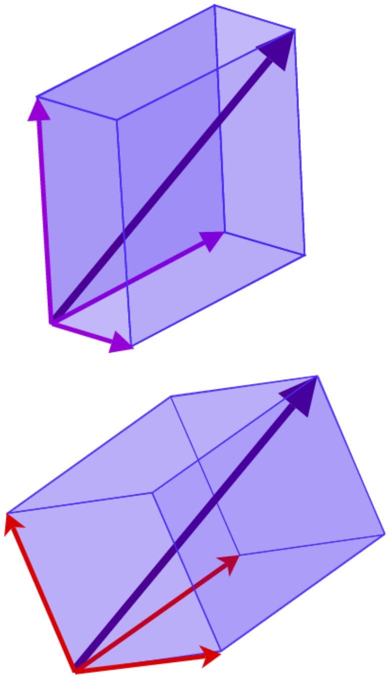


RBF networks

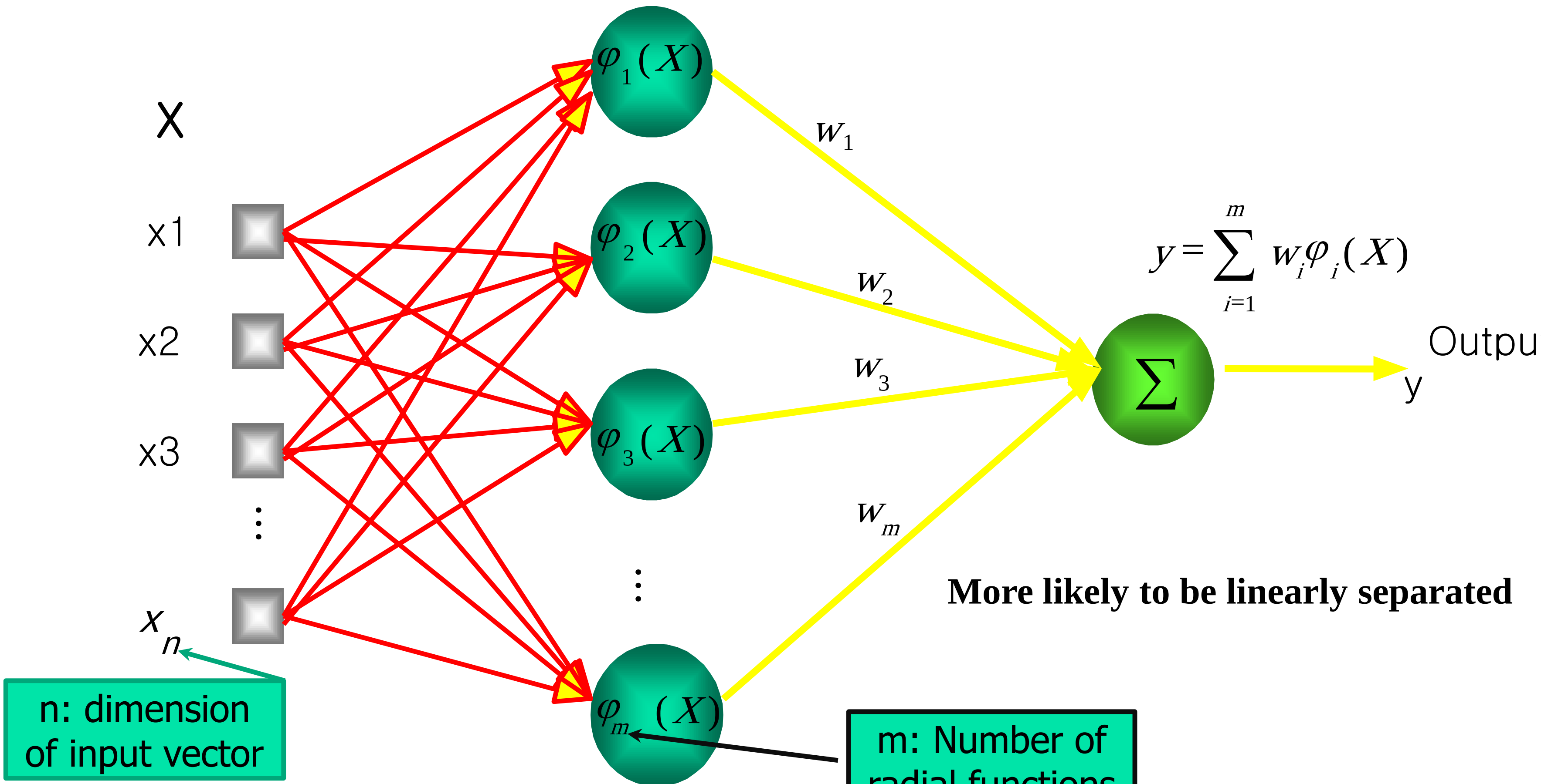
Definition :

Radial basis function (RBF) networks are a special class of single hidden-layer feed forward neural networks for application to problems of supervised learning.

The model '*f*' is expressed as a linear combination of a set of '*m*' *fixed* functions often called basis functions by analogy with the concept of a vector being composed of a linear combination of basis vectors.



RBFN: A 3-layer network



$$y = \sum_{i=1}^{m} w_i \varphi_i(X)$$

More likely to be linearly separated

Increased dimension: n->m

$\varphi_i(X)$: **non linear** function

RBFN: A 3-layer network

- Input layer
  - Source nodes that connect the network to its environment
- Hidden layer
  - Hidden units provide a set of basis function
  - High dimensionality
- Output layer
  - Linear combination of hidden functions

RBF networks

- Unlike most FF neural networks, the connection weights between the input layer and the neuron units of the hidden layer for an RBFN are all equal to unity.

- Each hidden neuron calculates a norm that represents the distance between the input to the network and the so-called position of the neuron (center). This is inserted into a radial activation function which calculates and outputs the activation of the neuron.

RBF parameters

A function is radial basis (**RBF**) if its output depends on the distance of the input from a given stored vector (a nonincreasing function).

Radial activation function

|| . ||
Usually, Euclidian distance

$$\varphi(\|x - c_j\|)$$

Non-linear function

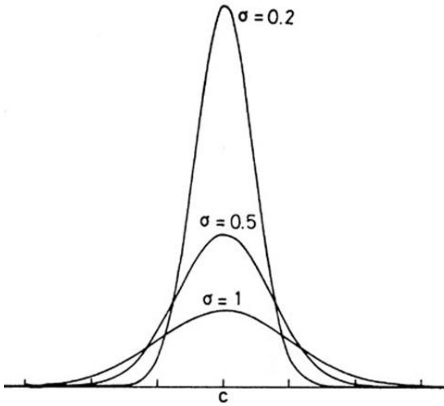
Center

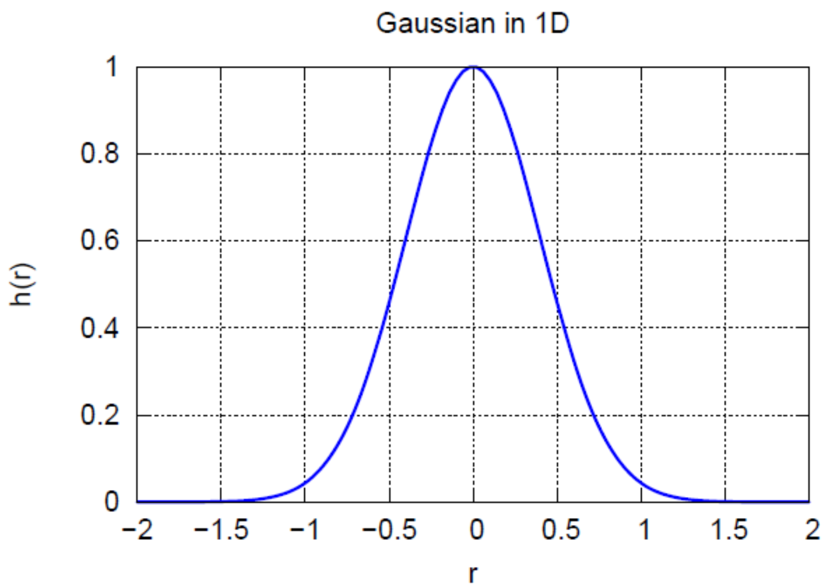
Input vector

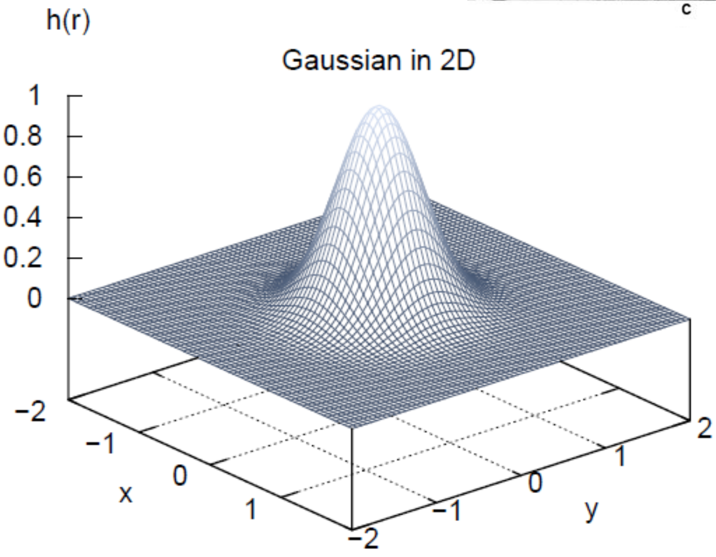
Radial activation function

A typical radial function is the **Gaussian** which in the case of a scalar input is

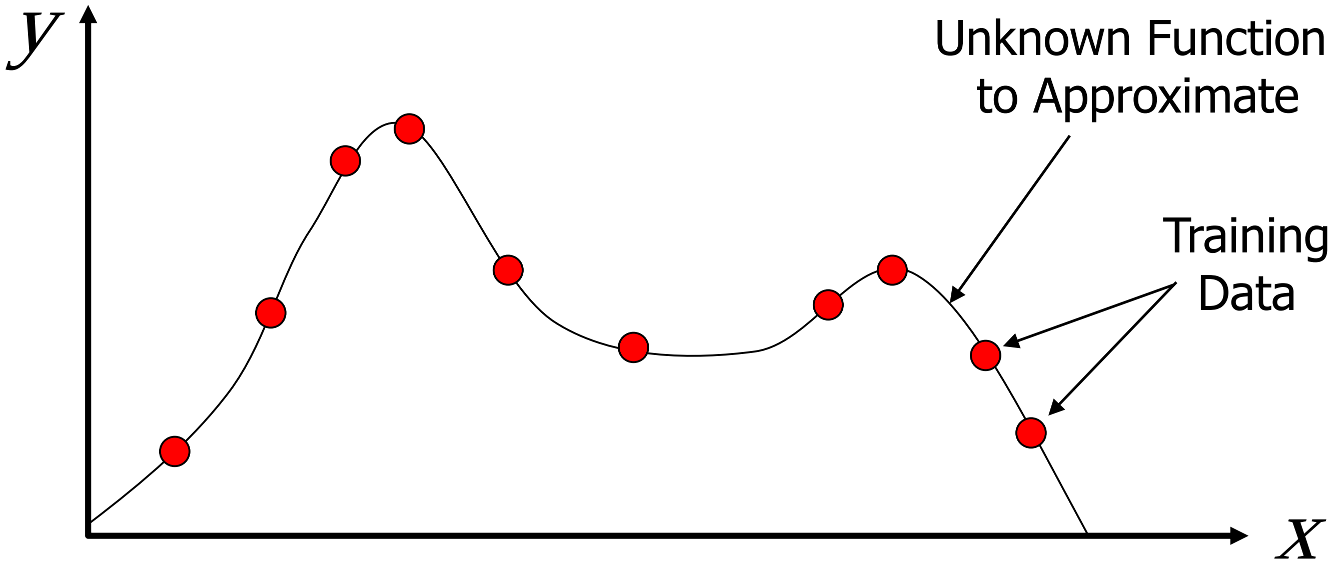
$$\varphi(x,c) = \exp\left[-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2\right]$$



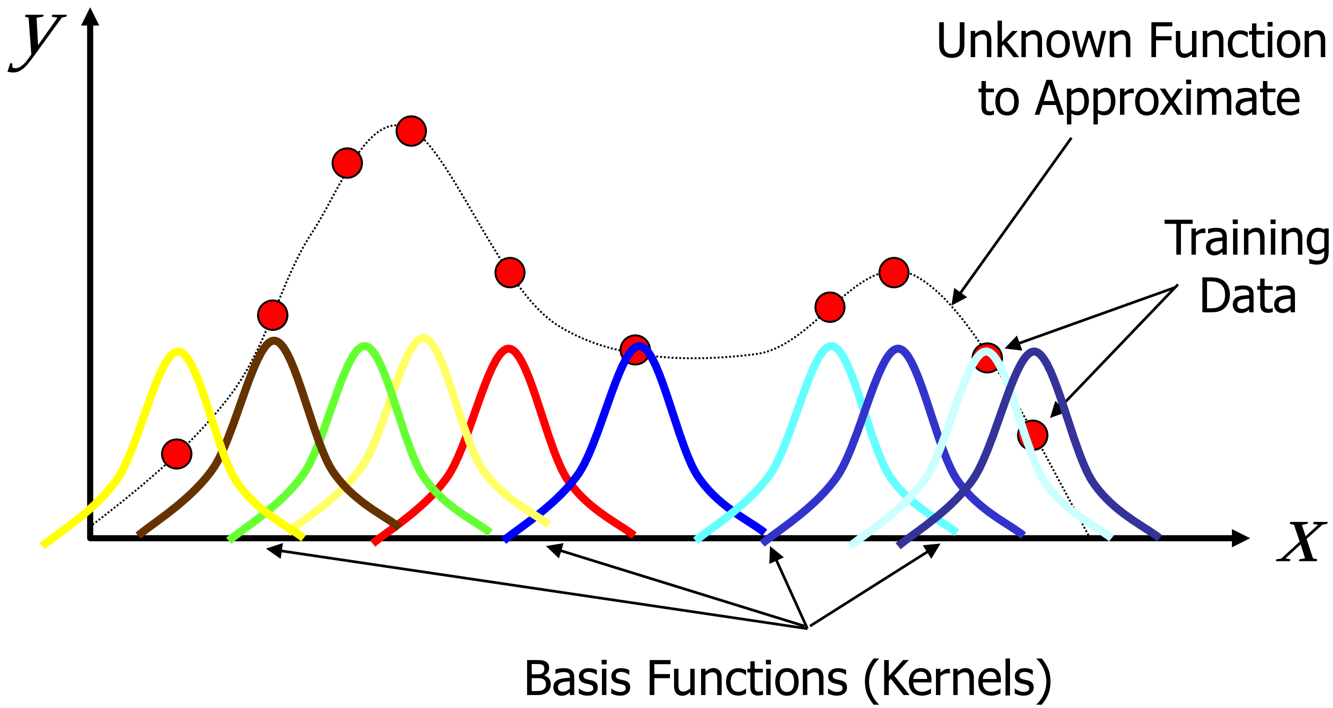




RBN First idea (Function approximation)

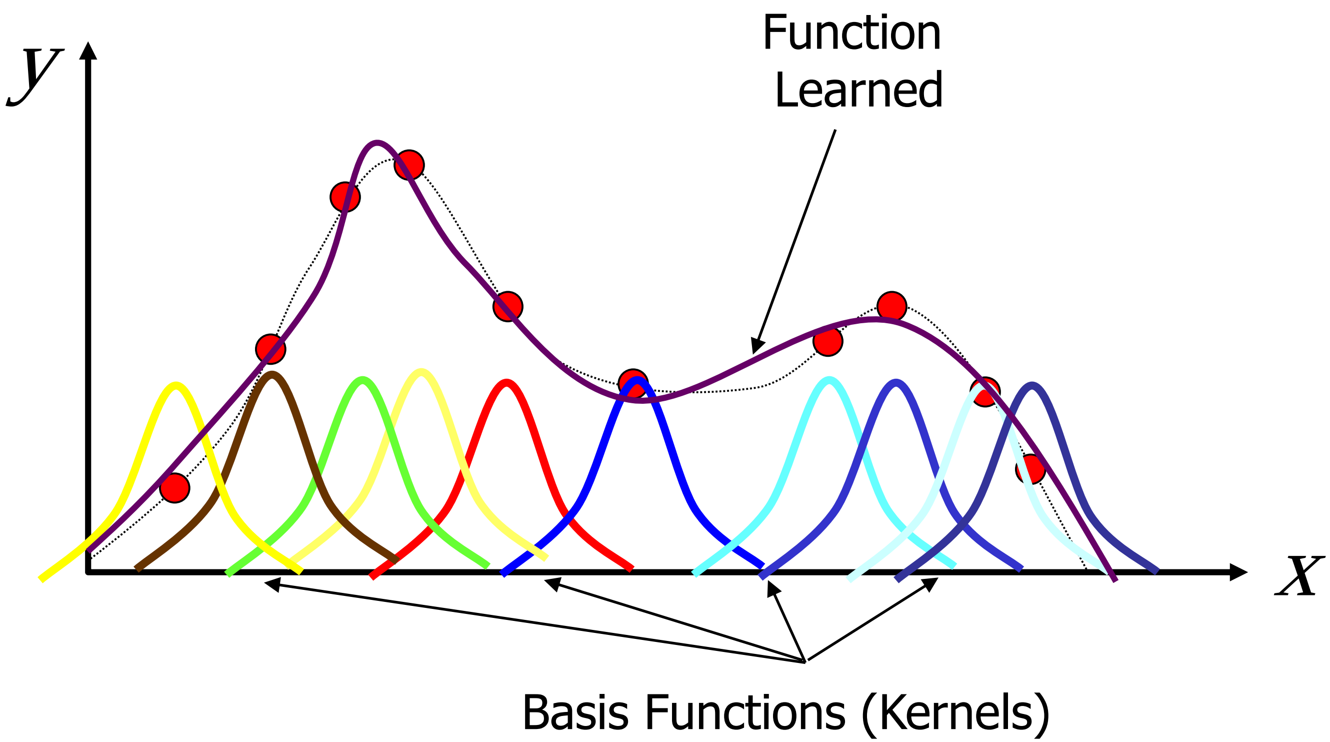


RBN First idea (Function approximation)



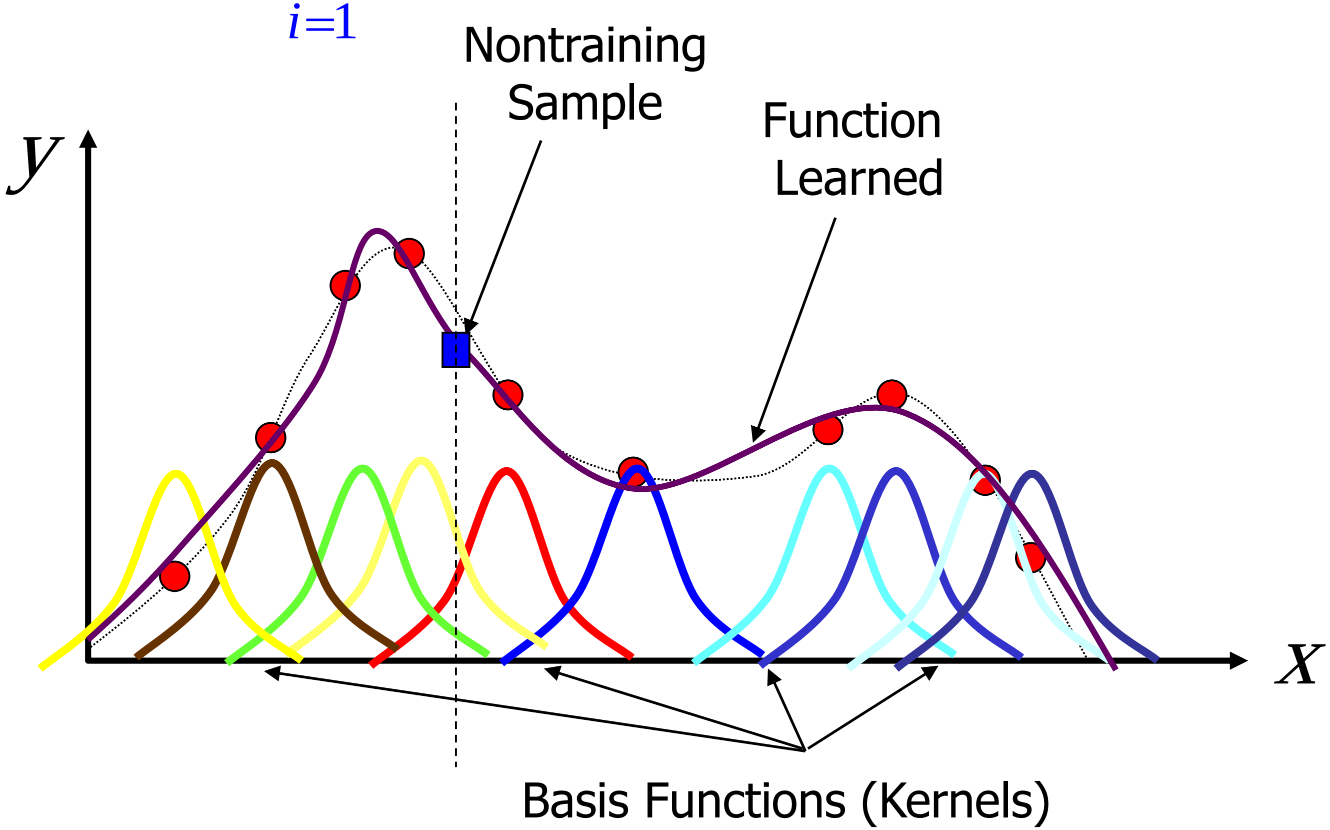
RBN First idea (Function approximation)

$$y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

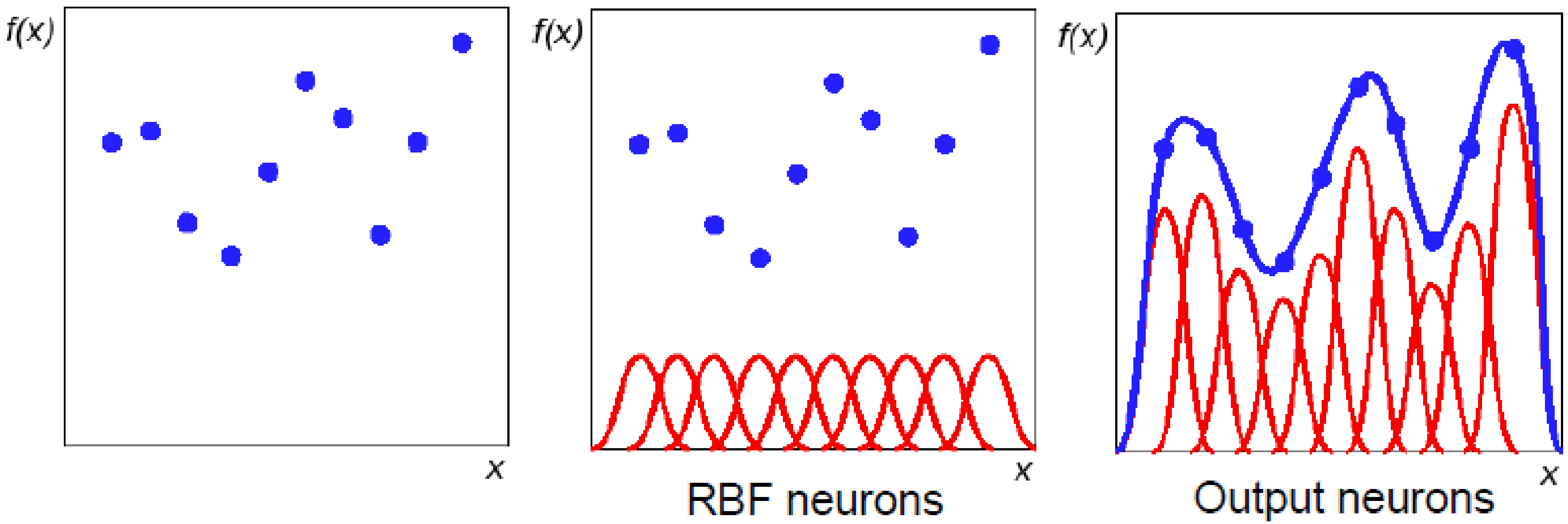


RBN First idea (Function approximation)

$$y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

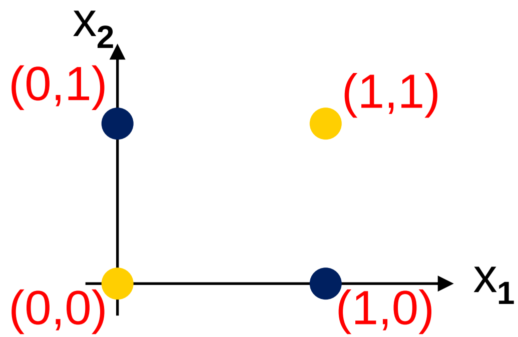


RBN First idea (Function approximation)

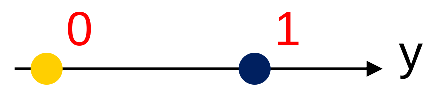


XOR Problem

- 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

XOR Problem

- **In the feature (hidden layer) space:**

$$\varphi_1(\| x - t_1 \|) = e^{-\|x - t_1\|^2}$$

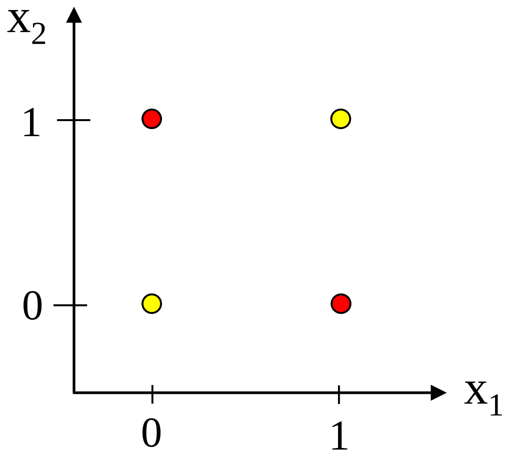
$$\varphi_2(\| x - t_2 \|) = e^{-\|x - t_2\|^2}$$

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

$$t_1 = (1,1) \text{ and } t_2 = (0,0)$$

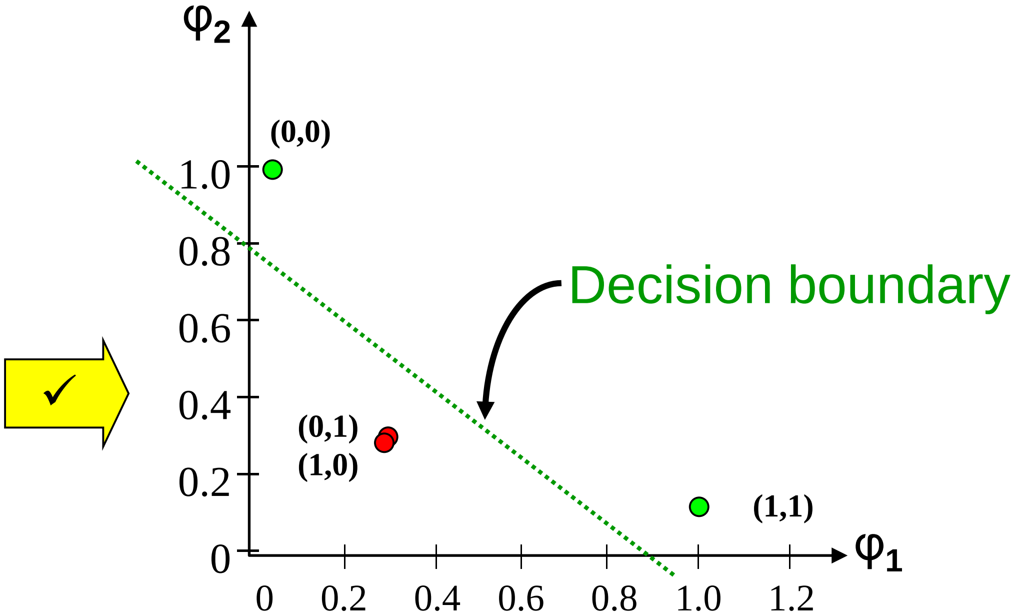
- **When mapped into the feature space < φ_1 , φ_2 > (hidden layer), C1 and C2 become *linearly separable.* So a linear classifier with φ_1(x) and φ_2(x) as inputs can be used to solve the XOR problem.**

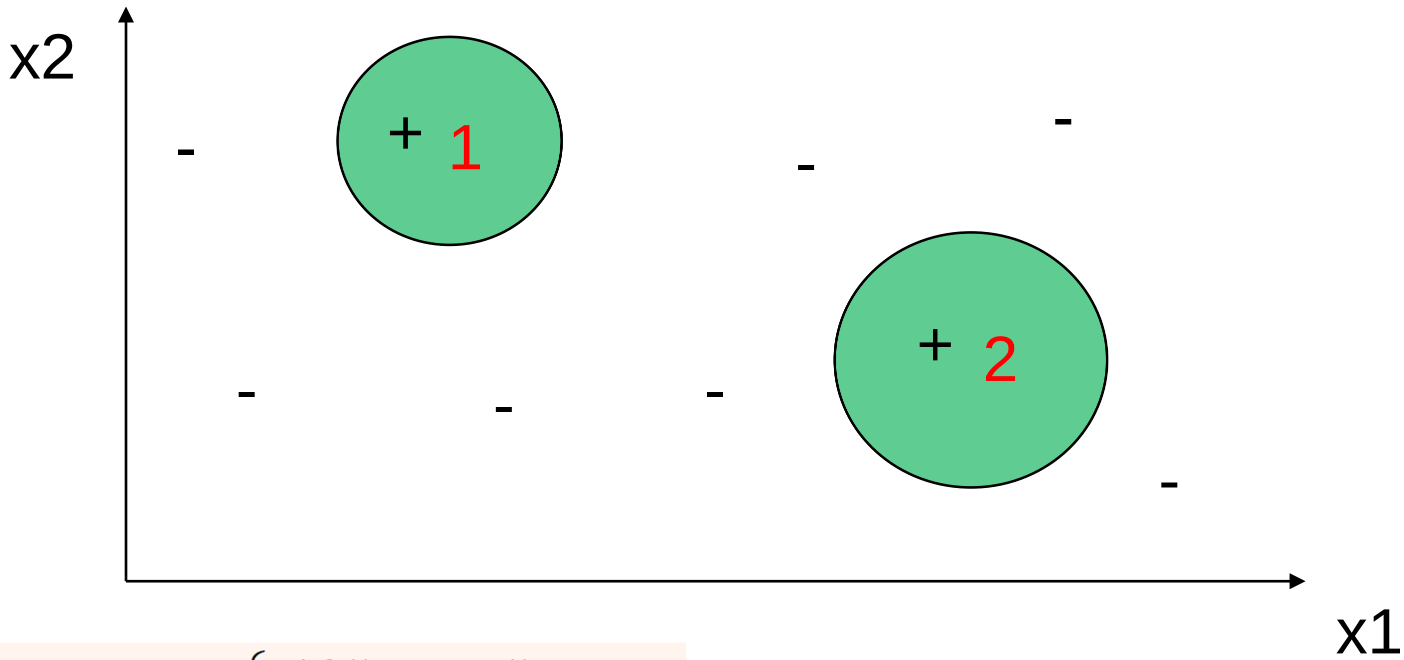
XOR Problem



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

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





$$\varphi_1(\| x-t_1 \|)=\begin{cases} 1 \text{ if } \| x-t_1 \| <= r1 \\ 0 \text{ if } \| x-t_1 \| > r1 \end{cases}$$

$$\varphi_2(\| x-t_2 \|)=\begin{cases} 1 \text{ if } \| x-t_2 \| <= r2 \\ 0 \text{ if } \| x-t_2 \| > r2 \end{cases}$$

t1 و t2 are centers of the circles

Learning Algorithms

- **Parameters** to be learnt are:
  - Centers
  - Spreads
  - Weights

- Different learning algorithms

Learning method1

- **Centers** are chosen randomly from the training set (can be equal to total training set)

- **Spreads** are chosen by **normalization**:

$$\sigma = \frac{\text{Maximum distance between any 2 centers}}{\sqrt{\text{number of centers}}} = \frac{d_{max}}{\sqrt{m_1}}$$

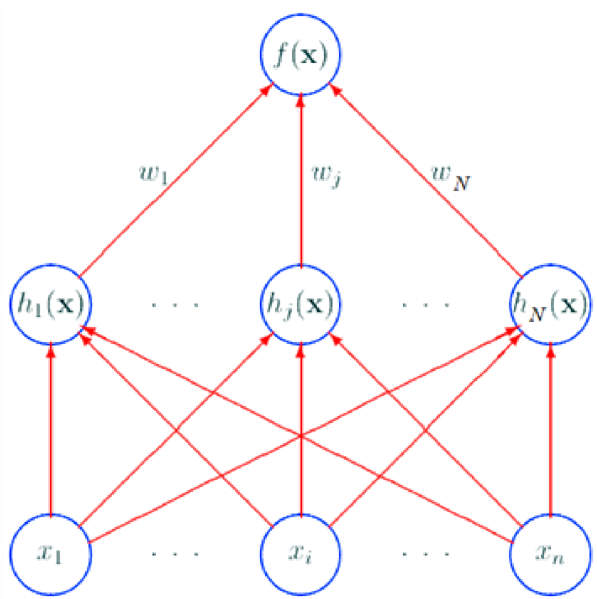
$$\varphi_i\left(\|x - t_i\|^2\right) = \exp\left(-\frac{m_1}{d_{max}^2}\|x - t_i\|^2\right)$$

$$i \in [1, m_1]$$

Learning method1

- **Weights** are found by means of **pseudo-inverse method**

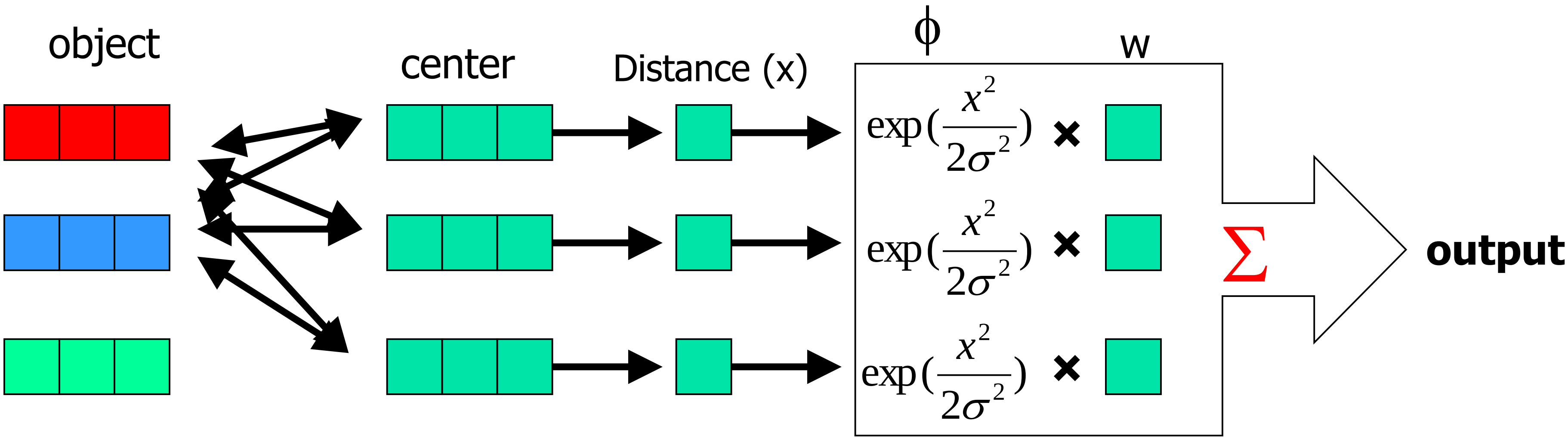
$$y_i = \sum_{k=1}^{N} w_k \varphi(\|x_i - x_k\|) \quad \Rightarrow \quad \mathbf{y} = \boldsymbol{\varphi}\,\mathbf{w}$$



$$\begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_N \end{bmatrix} = \begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdot & \cdot & \cdot & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdot & \cdot & \cdot & \varphi_{2N} \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ \cdot & & & & \cdot & \\ \varphi_{N1} & \varphi_{N2} & \cdot & \cdot & \cdot & \varphi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \cdot \\ \cdot \\ \cdot \\ w_N \end{bmatrix}$$

$$\mathbf{w} = \boldsymbol{\varphi}^{-1}\,\mathbf{y}$$

Learning method1

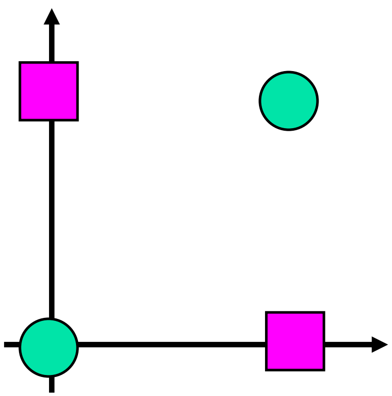


 # Hidd. nodes = # objects

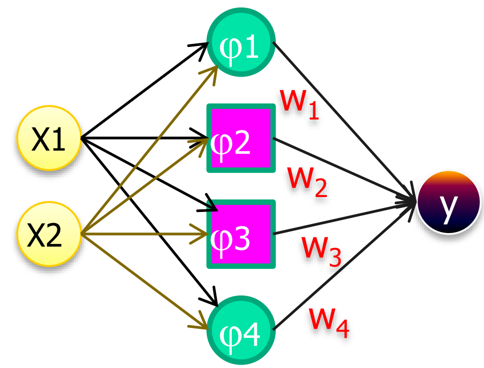
↓

Exact Fitting

Learning method1 (XOR example)



x	y
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

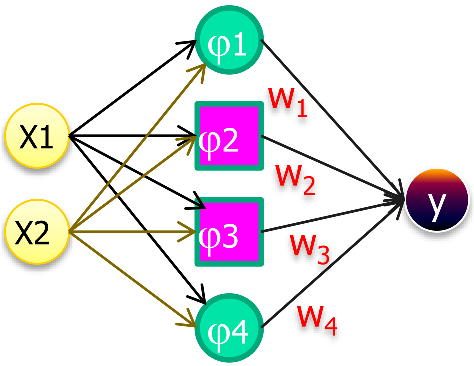


$$\varphi_{ik} = \varphi(\|\mathbf{x}_i - \mathbf{x}_k\|) = \exp(-0.5\|\mathbf{x}_i - \mathbf{x}_k\|^2)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} \varphi_{11} & \varphi_{12} & \varphi_{13} & \varphi_{14} \\ \varphi_{21} & \varphi_{22} & \varphi_{23} & \varphi_{24} \\ \varphi_{31} & \varphi_{32} & \varphi_{33} & \varphi_{34} \\ \varphi_{41} & \varphi_{42} & \varphi_{43} & \varphi_{44} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} e^{0} & e^{-0.5} & e^{-0.5} & e^{-1} \\ e^{-0.5} & e^{0} & e^{-1} & e^{-0.5} \\ e^{-0.5} & e^{-1} & e^{0} & e^{-0.5} \\ e^{-1} & e^{-0.5} & e^{-0.5} & e^{0} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \Rightarrow \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} -3.03 \\ 3.42 \\ 3.42 \\ -3.03 \end{bmatrix}$$

Learning method1 (XOR example)

x	y
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0



$$f(x_1,x_2)=\sum_{k=1}^{4} w_k\varphi_{ik}=\sqrt{x_1^2+x_2^2}-\frac{1}{\sqrt{2}}\sqrt{x_1^2+(x_2-1)^2}-\frac{1}{\sqrt{2}}\sqrt{(x_1-1)^2+x_2^2}+\sqrt{(x_1-1)^2+(x_2-1)^2}$$

$$f(x_1,x_2)=\sum_{k=1}^{4} w_k\varphi_{ik}=-3.0359\exp\left(-\frac{x_1^2+x_2^2}{2}\right)+3.4233\exp\left(-\frac{x_1^2+(x_2-1)^2}{2}\right)$$
$$+3.4233\exp\left(-\frac{(x_1-1)^2+x_2^2}{2}\right)-3.0359\exp\left(-\frac{(x_1-1)^2+(x_2-1)^2}{2}\right)$$

Learning method2

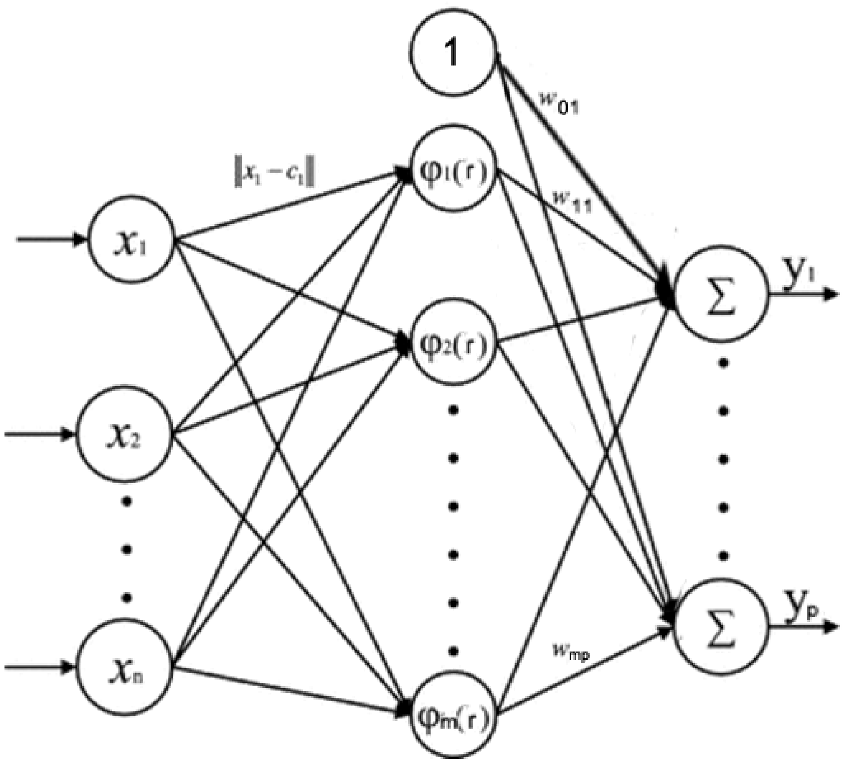
- **Hybrid Learning Process:**
  - **Self-organized learning stage** for finding the centers
  - Spreads chosen by normalization
  - **Supervised learning stage** for finding the weights, using LMS algorithm

Centers are obtained from unsupervised learning (clustering).

Spreads are obtained as variances of clusters, **w** are obtained through LMS algorithm. Clustering (k-means) and LMS are iterative. This is the most commonly used procedure. Typically provides good results.

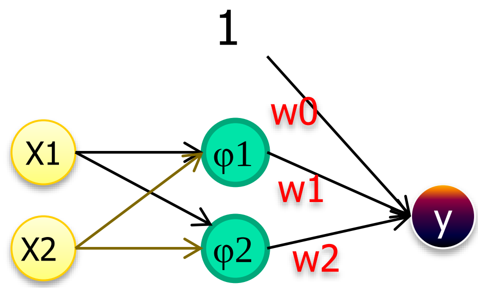
RBF structure with learning method 2

- Network structure
  - n input neurons
  - m RBF neurons
  - p output neurons



Learning method2 (XOR example)

x	y
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0



$$\boldsymbol{\mu}_1 = [1,1] \qquad \boldsymbol{\mu}_1 = [0,0]$$

$$d_{\max} = \|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\| = \sqrt{2} \Rightarrow \sigma = \frac{d_{\max}}{\sqrt{2m}} = \frac{1}{\sqrt{2}}$$

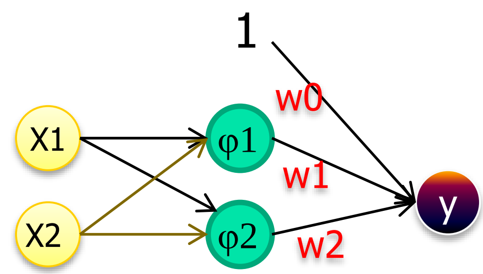
$$\varphi_{ij}(\mathbf{x}) = \exp\left[-\frac{\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right] = \exp\left(-\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2\right)$$

$$\varphi_1 = \exp\left(-\|\mathbf{x} - \boldsymbol{\mu}_1\|^2\right) \qquad \varphi_2 = \exp\left(-\|\mathbf{x} - \boldsymbol{\mu}_2\|^2\right) \qquad \mathbf{x} = [x_1, x_2]$$

$$y = w_0 + w_1\varphi_1 + w_2\varphi_2$$

Learning method2 (XOR example)

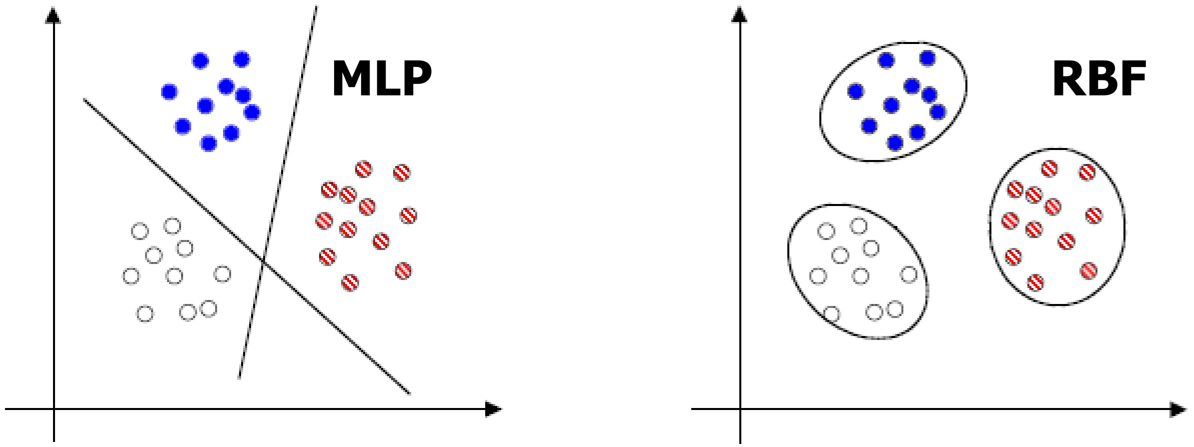
x	y
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

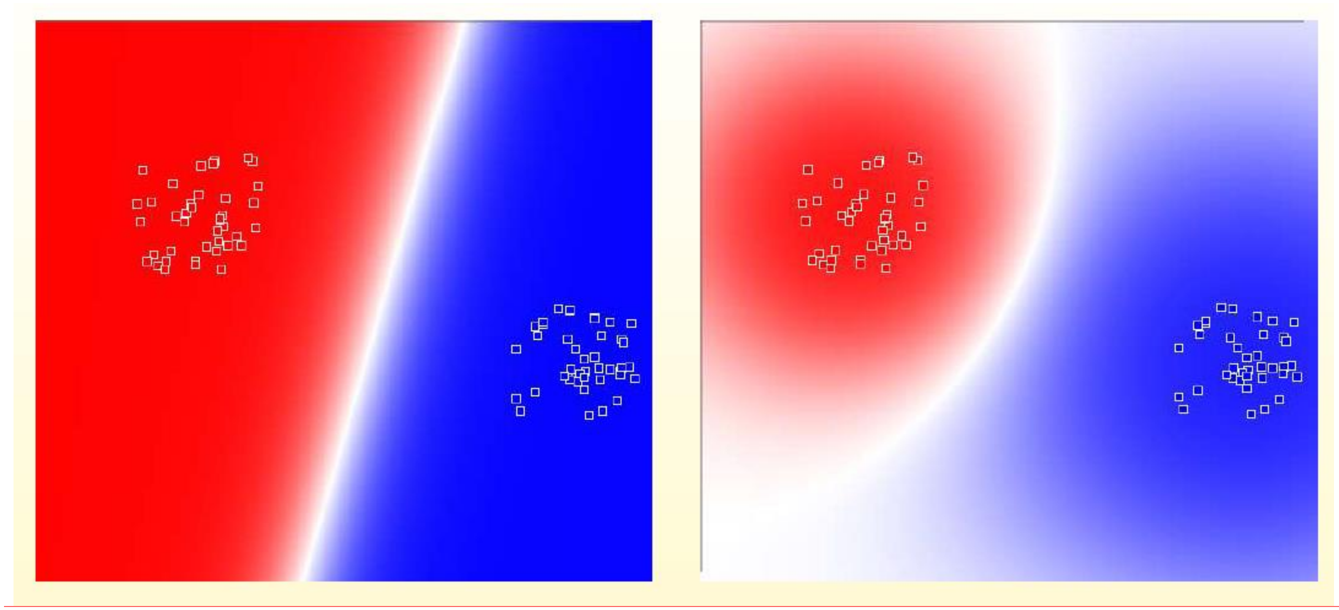


$$\begin{cases} y_1 = 0 = w_0 + w_1 \exp\left(-\left\|[0,0]-[0,0]\right\|^2\right) + w_2 \exp\left(-\left\|[0,0]-[1,1]\right\|^2\right) = w_0 + w_1 + 0.1353 w_2 \\ y_2 = 1 = w_0 + w_1 \exp\left(-\left\|[0,1]-[0,0]\right\|^2\right) + w_2 \exp\left(-\left\|[0,1]-[1,1]\right\|^2\right) = w_0 + 0.3679 w_1 + 0.3679 w_2 \\ y_2 = 1 = w_0 + w_1 \exp\left(-\left\|[1,0]-[0,0]\right\|^2\right) + w_2 \exp\left(-\left\|[1,0]-[1,1]\right\|^2\right) = w_0 + 0.3679 w_1 + 0.3679 w_2 \\ y_1 = 0 = w_0 + w_1 \exp\left(-\left\|[1,1]-[0,0]\right\|^2\right) + w_2 \exp\left(-\left\|[1,1]-[1,1]\right\|^2\right) = w_0 + 0.1353 w_1 + w_2 \end{cases}$$

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0.1353 \\ 1 & 0.3679 & 0.3679 \\ 1 & 0.3679 & 0.3679 \\ 1 & 0.1353 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \Rightarrow \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 2.8404 \\ -2.5018 \\ -2.5018 \end{bmatrix}$$

Comparison with MLP





Comparison with MLP

- Both are examples of *non-linear layered feed-forward* networks.
- Both are *universal approximators*.
- Hidden layers:
  - RBF networks have one *single* hidden layer.
  - MLP networks may have *more* hidden layers.

Comparison with MLP

- **Neuron Models:**
  - The computation nodes in the hidden layer of a RBF network are *different*. They serve a different purpose from those in the output layer.
  - Typically computation nodes of MLP in a hidden or output layer share a *common neuron model*.

- **Linearity:**
  - The hidden layer of RBF is non-linear, the output layer of RBF is linear.
  - Hidden and output layers of MLP are usually non-linear.

Comparison with MLP

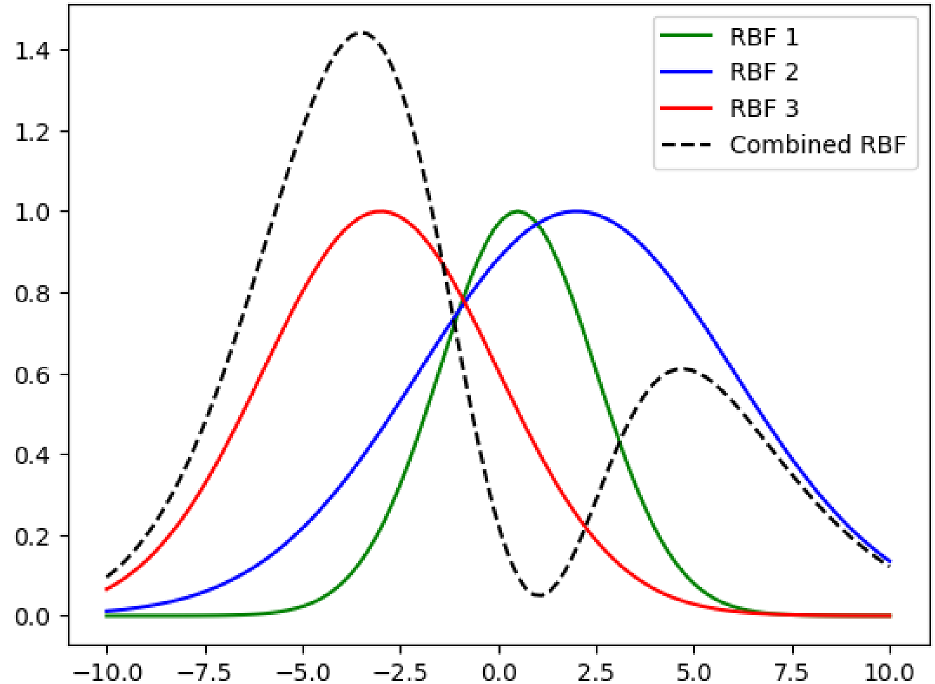
- **Activation functions:**
  - The argument of activation function of each hidden unit in a RBF NN computes the Euclidean distance between input vector and the center of that unit.
  - The argument of the activation function of each hidden unit in a MLP computes the inner product of input vector and the synaptic weight vector of that unit.
- **Approximations:**
  - RBF NN using Gaussian functions construct local approximations to non-linear I/O mapping.
  - MLP NN construct global approximations to non-linear I/O mapping.

RBF in python

```
import matplotlib.pyplot as plt
import numpy as np

def rbf(x, c, s):
    return np.exp(-1 / 2*((x-c)/s) **2 )
# 100 linearly spaced numbers
x = np.linspace(-10,10,100)

y1 = rbf(x, 0.5, 2)
y2 = rbf(x, 2, 4)
y3 = rbf(x, -3, 3)
y= -1.4*y1 +.9*y2+ 1.3*y3
plt.plot(x, y1, 'g', label='RBF 1')
plt.plot(x, y2, 'b', label='RBF 2')
plt.plot(x, y3, 'r', label='RBF 3')
plt.plot(x, y, 'k--', label='Combir
plt.legend()
plt.show()
```




Questions