SysEng 5212 /EE 5370
Introduction to Neural Networks and Applications

Lecture 7: Radial Basis Functions I

Cihan H Dagli, PhD

Professor of Engineering Management and Systems Engineering
Professor of Electrical and Computer Engineering
Founder and Director of Systems Engineering Graduate Program

dagli@mst..edu

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY
Rolla, Missouri, U.S.A.

Volunteers Tally

- Meryem Deniz
- Arnold Anthony Fernandes (1)

Volunteers Tally

- Abdul Ghafoor (1)
- Jonathan Griffin
- Prashanth Haridass
- David Hartmann
- George Holmes (1)

Volunteers Tally

- Zhangli Hu (1)
- Md Monirul Islam
- Garrett Kaiser
- Yogesh Lad
- Joshua Liao

Volunteers Tally

- Prasanth Mariappan
- Niklas Melton
- Dana Payonk (1)
- Sasha Petrenko (2)
- Timothy Sheppard

Volunteers Tally

- Connor Sprague
- Benjamin Thomas (1)
- Xiahan Yang
- Raghu Yelugam
- Nicholas York

Lecture outline

- Midterm exam is on 3/14/2017
  - Open-book + open notes + open Internet { Don't forget to credit your sources}
  - Two and half hours
  - Turn in the completed exam via CANVAS or submit to me in class
  - Must sign and attach honor code -exam will not be graded without it

Exam I Review

1. Do you understand the nonlinear neuron (perceptron) model? Can you create a weight matrix and bias and initialize them randomly? Can you calculate the activation potential?
2. Why do we use the bias value?
3. Can you classify 2 vectors correctly using the perceptron model?
4. Do you know all the different types of activation functions? Do you know their respective Matlab functions? What is the purpose of the squashing function?
5. Do understand the Least Mean Square (LMS) learning algorithm? Can you use the LMS algorithm to train a linear element?

Exam I Review

6. Do you know the preceptron learning rule and the expression for calculating the error? Given input and target vectors can you use the perceptron for classification? Can you calculate the mean square error and plot it?

7. Can you implement the perceptron with a sigmoid activation function?

8. Can you implement a perceptron or a linear neuron with LMS learning rules?

9. Do you understand error surfaces and weight behavior?

10. Implement the standard backpropagation learning algorithm without using the MATLAB toolbox. Use it to approximate nonlinear functional mappings.

Exam I Review

11. Use Nguyen and Widrow's initialization algorithm to initialize the synaptic weights of the multi-layer perceptron neural network.

12. Implement the backpropagation learning algorithm with momentum updating without using the MATLAB toolbox. Use it to approximate nonlinear functional mappings.

13. Implement the backpropagation learning algorithm using batch-updating of weights.

14. Implement the search-then-converge method of adjusting the learning rate to speed up backpropagation learning.

15. Use batch updating with variable learning rate to increase the convergence speed of the backpropagation algorithm.

Exam I Review

16. Use the MATLAB toolbox to implement the standard backpropagation algorithm and all other variations of the backpropagation algorithm.

17. Do you remember the heuristics for improving the performance of the backpropagation algorithm?

18. Can you provide a simple answer for the following questions.

a. Why should we avoid initial weights with large values?

b. Why should we normalize and randomize the inputs?

c. Why should we prefer bipolar inputs to binary ones?

d. In a feedforward network trained with backpropagation, when should we use a linear output layer and when should we use a nonlinear output layer?

Exam I Review

e. What is the learning rate and how does it affect convergence?

f. What is the momentum constant also called forgetting factor? How does it affect the trajectory of convergence?

g. Can we train a feedforward network with the hardlimit activation function using backpropagation? Why or why not?

h. Differences between batch learning and incremental learning.

i. What is overfitting? How can we avoid it?

Exam I Review

19. Train a radial basis function neural network with fixed centers and use it to approximate a nonlinear functional mapping. Do this without using the MATLAB toolbox.

20. Train a radial basis function neural network using the stochastic gradient approach. Do this without using the MATLAB toolbox.

21. Design an RBF NN using the MATLAB toolbox.

22. What is the purpose of the hidden neurons in an RBF?

Lecture outline

1 RBF Introduction

2 Cover's Theorem

3 Separability of patterns

4 RBF Neuron

5 XOR example

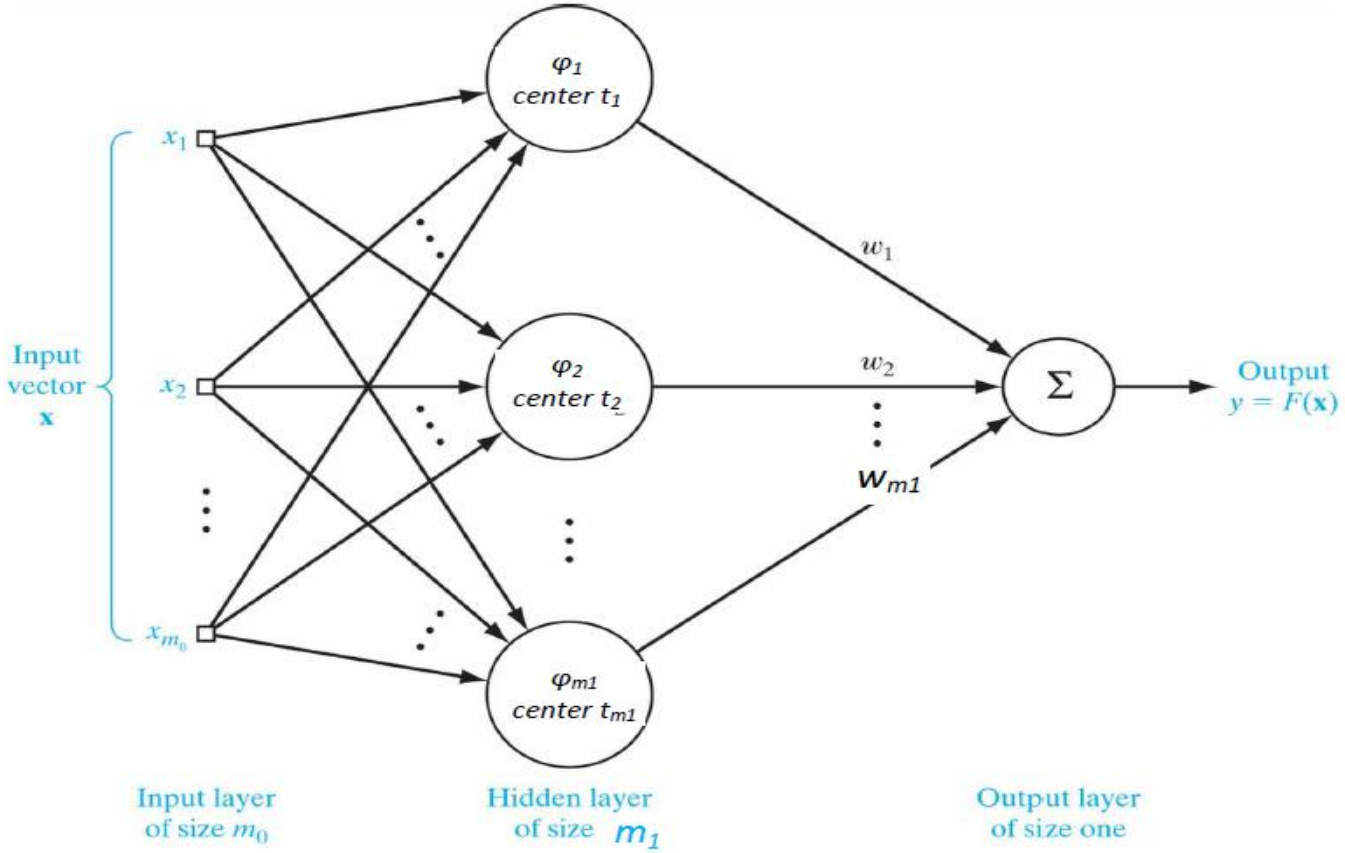
6 Learning algorithms

Radial-Basis Function Networks

- RBFN represent an alternative approach in designing supervised neural networks
- NN design as a curve fitting problem
- Goal is to find a surface in multidimensional space that provides a best fit to the training data
- Best fit is determined statistically
- Generalization is done by interpolating on the multidimensional surface
- Hidden neurons use functions that act as a basis set for transforming the inputs into the hidden higher dimensional space
  - These functions are called radial-basis functions
- Radial-basis function, Nonlinear transformation followed by linear transformation

MATLAB: y = purelin(y)

Architecture of RBF



Mathematical Justification

Cover's theorem

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

Simply put,

- Use a nonlinear function to map input samples from the input space to a hidden space that has higher dimension than the input space.
- In the higher dimension use simple algorithms to find the separating hyperplane

Separability of Patterns

- Consider N input samples, $X = x1, x2, \ldots, xN$
- Each input sample is m_0-dimensional, and belongs to one of two classes, X1 and X2, forming a *dichotomy*.
- This binary partition of the points is said to be separable with respect to a family of surfaces if there exists a surface in the family that separates the points in the class X1 from those in the class X2
- For each $x \in X$ define a vector made of real-valued functions $\{\emptyset(x)|i = 1,2, \ldots, m_1\}$ as shown,

$$\emptyset(x) = [\emptyset_1(x) + \emptyset_2(x) + \ldots\ldots + \emptyset_{m_1}(x)]$$

This vector maps the m_0-dimensional input to the m_1-dimensional hidden space or feature space. The functions $\emptyset(x)$ are called the hidden functions in keeping with the hidden neurons of feed-forward networks.

Separability of Patterns 2

A dichotomy $\mathcal{X}_1, \mathcal{X}_2$ is ϕ-separable if there exists an m_1-dimensional vector $\mathbf{w}$,

$$\mathbf{w}\phi(x) > 0, \ x \in \mathcal{X}_\infty$$
$$\mathbf{w}\phi(x) < 0, \ x \in \mathcal{X}_\in$$

where, $\mathbf{w}\phi(x) < 0$ is the separating hyperplane.

Suppose that all possible dichotomies of $\mathcal{X} = \{x_i\}_{i=1}^N$ are equiprobable.

Probability that any random partition of $\mathcal{X}_1, \mathcal{X}_2$ is ϕ-separable,

$$P(N, m_1) = \left(\frac{1}{2}\right)^{N-1} \sum_{m=0}^{m_1-1} \binom{N-1}{m}$$

Separability of Patterns

The higher we make the dimension m_1 of the hidden space, the higher the probability $P(N, m_1)$ will be.

- $P(N, m_1)$ tends to 1 as the dimensions of the hidden space increase.
- Means that it is more likely that the classes will become separable
- More functions in the hidden space or feature space imply greater flexibility in classification.

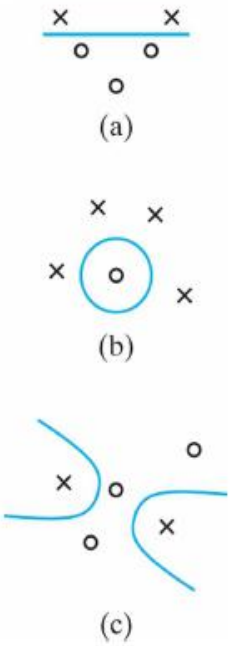
Cover's Theorem Key Ideas

1. Nonlinear formulation of the hidden function $\emptyset_i\ (x)$where x is the input vector.
2. High dimensionality of the hidden space compared to the input space; the dimensionality of the hidden space is determined by the number of hidden units.

In general,

1. A complex pattern-classification problem nonlinearly transformed into a higher dimension, is more likely to be linearly separable than in a low-dimensional space.
2. In some cases, simply transforming the input nonlinearly may be sufficient to produce linear separability.

Examples of Φ-Separable Patterns

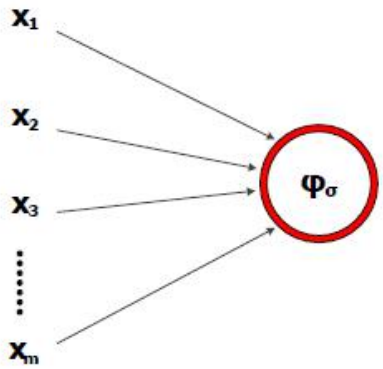


(a) Linearly separable (b) Spherically separable (c) Quadrically separable

Hidden Neuron of the RBF Network

- The hidden units use a radial basis function of the type,

$$\phi_\sigma(\|x - t\|^2)$$



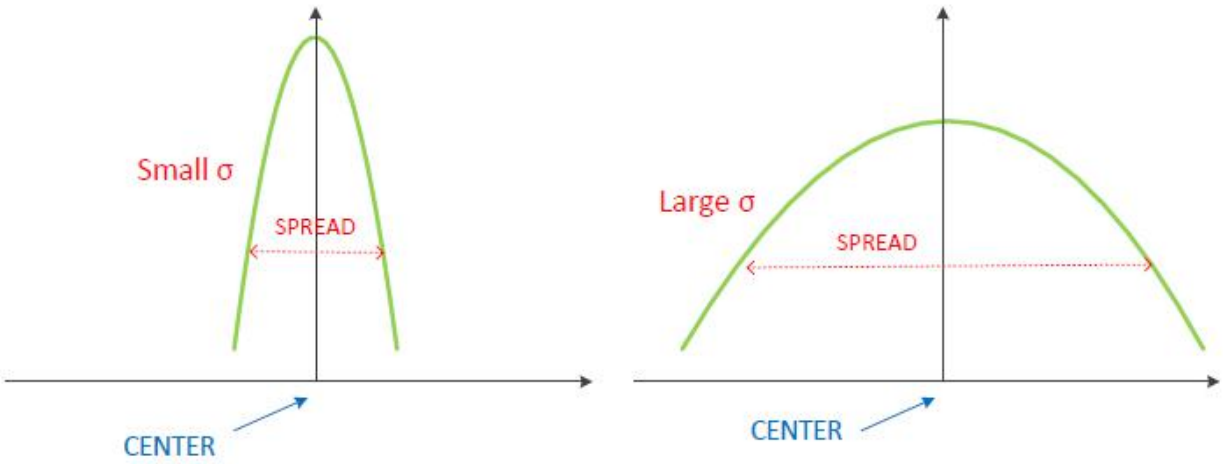
- where, x is the input, t is the center, and σ is called the spread.
- The output is determined by the distance of the input from this center.

Hidden Neuron Sensitivity

- The hidden neuron is more sensitive to data points closer to its center.
- The spread parameter σ governs the sensitivity.
- We can adjust the sensitivity by adjusting σ

Illustration of RBF Parameters

Consider a Gaussian radial-basis function as an example:



σ is a measure of the spread of the curve.

Types of Radial-Basis Functions

Some choices for $\phi(\cdot)$,

- $\phi(x) = x$, Linear function
- $\phi(x) = x^3$, Cubic approximation
- $\phi(x) = x,^2 \ln x$, thin-plate spline function
- $\phi(x) = \exp^{-x^2/2\sigma^2}$, Gaussian function
- $\phi(x) = \sqrt{x^2 + \sigma^2}$, Multiquadratic function
- $\phi(x) = \frac{1}{\sqrt{x^2+\sigma^2}}$, Inverse multiquadratic function

Output of the RBFN

- The RBFN consist of three layers, an input layer, a single layer of nonlinear hidden neurons, and the output layer. The output is calculated according to, The basic update step of the Newton method is,

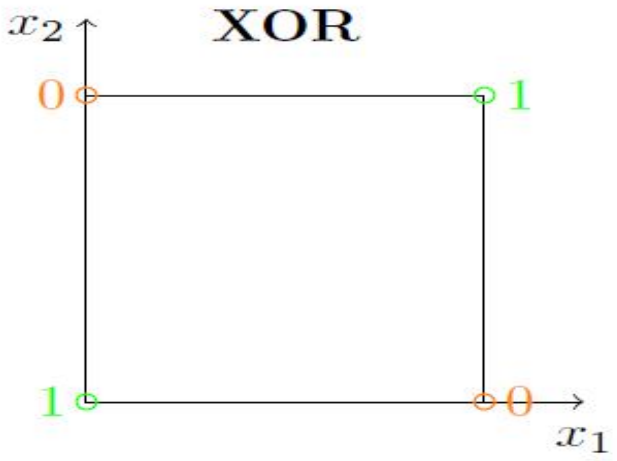
$$y_k = f_k(x) = \sum_{j=1}^{m_1} w_{kj}\phi_j(\|x - t_j\|_2) =$$

where k = 1,2,…..,m_2. w_{kj} are the weights in the output layer, and $\Phi(.)$ is the radial basis function.

Back to the XOR Problem

x_1	x_1	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Separation of the input space for the XOR function,



XOR Problem

Construct an RBF pattern classier that produces,

- 0 for inputs (1, 1) and (0, 0)
- 1 for inputs (0, 1) and (1, 0)

- Define a pair of Gaussian hidden functions

$$\phi_1(x) = e^{\|x-t_1\|^2}, \; t_1 = [1,1]^T$$

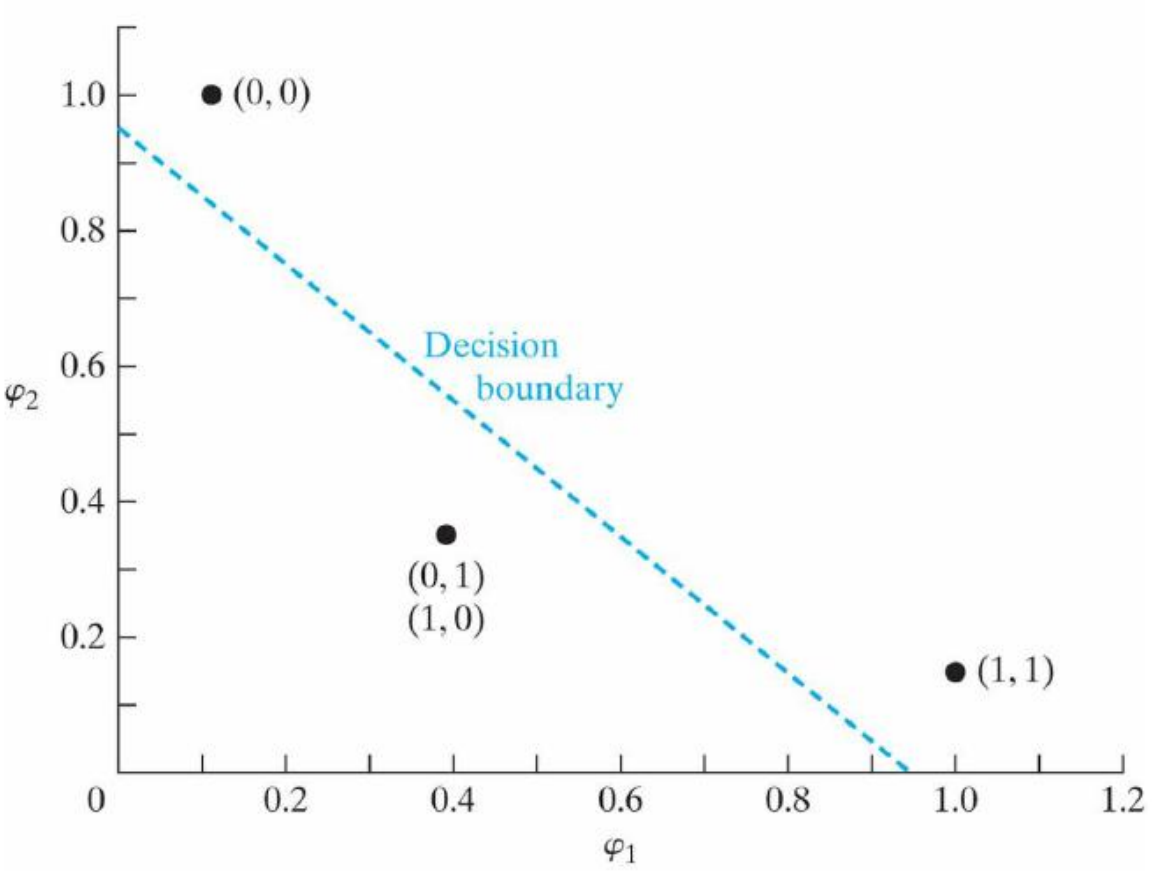
$$\phi_2(x) = e^{\|x-t_2\|^2}, \; t_2 = [0,0]^T$$

Finding the Hidden Functions for the Inputs

TABLE 5.1 Specification of the Hidden Functions for the XOR Problem of Example 1

Input Pattern $\mathbf{x}$	First Hidden Function $\varphi_1(\mathbf{x})$	Second Hidden Function $\varphi_2(\mathbf{x})$
(1,1)	1	0.1353
(0,1)	0.3678	0.3678
(0,0)	0.1353	1
(1,0)	0.3678	0.3678

The Φ_1 - Φ_2 Plane



Learning Algorithms

- The tunable parameters of the RBFN are,
  - centers
  - spreads
  - weights

Learning strategies,

- Training with fixed centers selected at random
- Stochastic gradient learning
- Hybrid learning

Training with Fixed Centers

- Centers are chosen randomly from the training data
- The hidden function used can be a Gaussian function

$$G(\|x - t_i\|^2) = exp\left(- \frac{m_1}{d_{max}^2} \|x - t_i\|^2 \right) \quad i = 1, 2, \ldots, m_1 \qquad (1)$$

- The spread of the radial-basis functions are chosen by normalization

$$\sigma = \frac{d_{max}}{\sqrt{m_1}} \qquad (2)$$

d_{max} is the max distance between any two centers and $\sqrt{m_1}$ the number of centers

Calculating the Weights

The output of the network is given by,

$$y_k = \sum_{j-1}^{m_1} w_{kj}\phi_j(\|x - t_j\|_2 \qquad (3)$$

Writing this in the vector-matrix form,

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m_2} \end{bmatrix} = \begin{bmatrix} \phi(\|x_1 - t_1\|_2 & \phi(\|x_1 - t_2\|_2 & \dots & \phi(\|x - t_{m_1}\|_2 \\ \phi(\|x_2 - t_1\|_2 & \phi(\|x_2 - t_2\|_2 & \dots & \phi(\|x - t_{m_1}\|_2 \\ \vdots & \vdots & \vdots & \vdots \\ \phi(\|x_{m_0} - t_1\|_2 & \phi(\|x_{m_0} - t_2\|_2 & \dots & \phi(\|x_{m_0} - t_{m_1}\|_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{m_1} \end{bmatrix}$$

$$\mathbf{y} = \phi\mathbf{w} \qquad (4)$$

Psuedoinverse for Computing Weights

Solving for w, $\mathrm{w} = \phi^{+}\mathrm{d}$

where, d is the desired response vector and Φ^{+} is the psuedoinverse of the nonlinear mapping matrix Φ.

- The psuedoinverse can be calculated as,

$$\phi^{+} = (\phi^{T}\phi)^{-1}\phi^{T}\mathrm{d} \qquad (5)$$

Steps for Training an RBFN with Fixed Centers

1. Choose RBF centers from the set of input samples. The number of centers selected should be large enough to ensure adequate sampling.
2. Calculate spread using equation (2)
3. Initialize weights to small random values
4. Calculate the network response using equation (4)
5. Solve for the weights using equation (5)

Learning Algorithm 2: Stochastic Gradient Approach

- The fixed center approach is very simple, but with a large dataset, the number of centers can become unmanageable.
- Very large network, computationally infeasible
- Stochastic gradient approach uses a gradient based approach to set the centers, spreads and weights.
- Being able to set all free parameters gives greater flexibility and improves performance
- Due to gradient descent approach, the algorithm can get stuck in local minima.

Steps for the Stochastic Gradient Approach

1. Choose the initial centers from the set of input vectors
2. Calculate the initial value of the spread parameter using equation (2)
3. Initialize the weights to small random values
4. Present each input vector to the network and compute the response,

$$y(n) = \sum_{j-1}^{m_1} w_j \phi(x(n), t_j, \sigma_j) \qquad (6)$$

Stochastic Gradient Learning

- Update the network parameters as shown,

$$w_j(n+1) = w_j(n) + \mu_w e_j(n)\phi(x(n), c_j, \sigma_j)$$

$$t_j(n+1) = t_j(n) + \mu_t \frac{e(n)w_j(n)}{\sigma_j^2(n)}\phi(x(n), t_j, \sigma_j)[x(n) - t_j(n)]$$

$$\sigma_j(n+1) = \sigma_j(n) + \mu_\sigma \frac{e(n)w_j(n)}{\sigma_j^3(n)}\phi(x(n), t_j, \sigma_j)\|x(n) - t_j(n)\|^2$$

where, $e(n) = y(n) - d(n)$ and μ_w, μ_t and μ_σ are the learning rate parameters.

- Stop if the network converges or go back to step 4.

Comparison with Multilayer Networks

- Both RBFN and MLP are nonlinear feedforward networks.
- RBFN have only one hidden layer and one set of weights to update from the output layer. MLP networks may have more hidden layers.
- Both networks are universal approximators
- The hidden neuron functions perform a nonlinear transformation of the input to a higher-dimensional space. The hidden neurons of a MLP act as feature detectors. The behavior is different.
- RBFN hidden layer is nonlinear and the output layer is linear. MLP hidden layers are nonlinear, but the output layer maybe linear or nonlinear.