Artificial Neural Networks
Lecture 5: Radial Basis Function Networks

Tudor Berariu
tudor.berariu@gmail.com

Faculty of Automatic Control and Computers
University Politehnica of Bucharest

Lecture : 11^{th} of November, 2015
Last Updated: 11^{th} of November, 2015

Today's Outline

Today's Outline

1. Radial Basis Function Networks
2. Training Radial Basis Function Networks
3. Comparison with Multi-Layer Feedforward Networks

Today's Outline

1. Radial Basis Function Networks
   - The Interpolation Problem
   - RBF Networks
   - RBF Networks for Classification
2. Training Radial Basis Function Networks
3. Comparison with Multi-Layer Feedforward Networks

Exact Interpolation

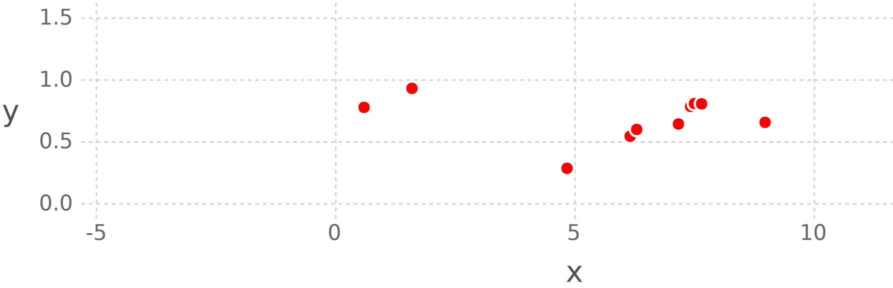


Figure: The exact interpolation problem

Exact Interpolation

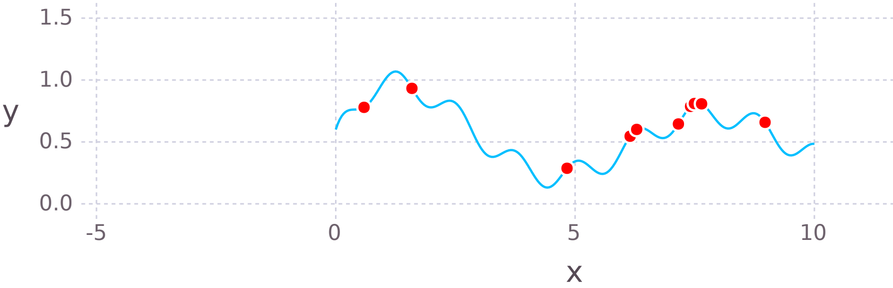


Figure: The exact interpolation problem

The Exact Interpolation Problem

Definition

Given a data set $\mathbf{X}$ that consists of N vectors $\mathbf{x}^{(n)}$ from a D-dimensional input space and N corresponding target vectors $\mathbf{t}^{(n)}$ in a K-dimensional output space, find K functions y_k such that:

$$y_k(\mathbf{x}^{(n)}) = t_k^{(n)}, \quad \forall k = 1, \ldots, K, \; n = 1, \ldots, N \tag{1}$$

Radial Basis Functions

- [Pow87] introduced N radial basis functions $\phi(\cdot)$ (one for each example)
- $\phi^{(n)}$ depends on $||\mathbf{x} - \mathbf{x}^{(n)}||$

Radial Basis Functions

- [Pow87] introduced N radial basis functions $\phi(\cdot)$ (one for each example)
- $\phi^{(n)}$ depends on $||\mathbf{x} - \mathbf{x}^{(n)}||$
- the output of the function is a linear combination of the basis functions:

$$y_k(\mathbf{x}) = \sum_{n=1}^{N} w_{kn}\phi(||\mathbf{x} - \mathbf{x}^{(n)}||) \tag{2}$$

Functions in matrix form

Formula 2

$$y_k(\mathbf{x}) = \sum_{n=1}^{N} w_{kn}\phi(||\mathbf{x} - \mathbf{x}^{(n)}||)$$

in matrix form looks like this:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1N} \\ w_{21} & w_{22} & \dots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K1} & w_{K2} & \dots & w_{KN} \end{bmatrix} \begin{bmatrix} \phi(||\mathbf{x} - \mathbf{x}^{(1)}||) \\ \phi(||\mathbf{x} - \mathbf{x}^{(2)}||) \\ \vdots \\ \phi(||\mathbf{x} - \mathbf{x}^{(N)}||) \end{bmatrix} \tag{3}$$

Interpolation condition

Because we need a perfect mapping (Formula 1):

$$y_k(\mathbf{x}^{(n)}) = t_k^{(n)}, \quad \forall k = 1, \dots, K,\ n = 1, \dots, N$$

$$\begin{bmatrix} w_{11} & w_{12} & \dots & w_{1N} \\ w_{21} & w_{22} & \dots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K1} & w_{K2} & \dots & w_{KN} \end{bmatrix} \begin{bmatrix} \phi(||\mathbf{x}^{(1)} - \mathbf{x}^{(1)}||) & \dots & \phi(||\mathbf{x}^{(N)} - \mathbf{x}^{(1)}||) \\ \phi(||\mathbf{x}^{(1)} - \mathbf{x}^{(2)}||) & \dots & \phi(||\mathbf{x}^{(N)} - \mathbf{x}^{(2)}||) \\ \vdots & \ddots & \vdots \\ \phi(||\mathbf{x}^{(1)} - \mathbf{x}^{(N)}||) & \dots & \phi(||\mathbf{x}^{(N)} - \mathbf{x}^{(N)}||) \end{bmatrix} = \begin{bmatrix} t_1^{(1)} & t_1^{(2)} & \dots & t_1^{(N)} \\ t_2^{(1)} & t_2^{(2)} & \dots & t_2^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ t_K^{(1)} & t_K^{(2)} & \dots & t_K^{(N)} \end{bmatrix}$$

Finding **W**

$$\overbrace{\mathbf{W}}^{\text{size: } K \times N} \cdot \overbrace{\mathbf{\Phi}}^{\text{size: } N \times N} = \mathbf{T} \tag{4}$$

where $\mathbf{\Phi} = \left\{ \phi(||\mathbf{x}^{(i)} - \mathbf{x}^{(n)}||) \right\}_{1 \leq i,j \leq N}$.

If $\mathbf{\Phi}^{-1}$ exists, the direct expression for **W** is:

$$\mathbf{W} = \mathbf{T}\mathbf{\Phi}^{-1} \tag{5}$$

- Micchelli proved that for some functions $\phi(\cdot)$ if examples are different then $\mathbf{\Phi}$ is non-singular.

Basis functions

Localized basis functions:

- Guassian

$$\phi(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{6}$$

- Inverse multi-quadric function:

$$\phi(x) = (x^2 + \sigma^2)^{-\alpha}, \quad \alpha > 0 \tag{7}$$

Basis functions

Non-localized basis functions:

- Thin-plate spline function:

$$\phi(x) = x^2 \log(x) \tag{8}$$

- Multi-quadric function:

$$\phi(x) = (x^2 + \sigma^2)^{-\beta}, \quad 0 < \beta < 1 \tag{9}$$

- The cubic function:

$$\phi(x) = x^3 \tag{10}$$

Basis functions

Non-localized basis functions:

- Thin-plate spline function:

$$\phi(x) = x^2 \log(x) \tag{8}$$

- Multi-quadric function:

$$\phi(x) = (x^2 + \sigma^2)^{-\beta}, \quad 0 < \beta < 1 \tag{9}$$

- The cubic function:

$$\phi(x) = x^3 \tag{10}$$

anyway, ... we are almost always using the Gaussian

Example

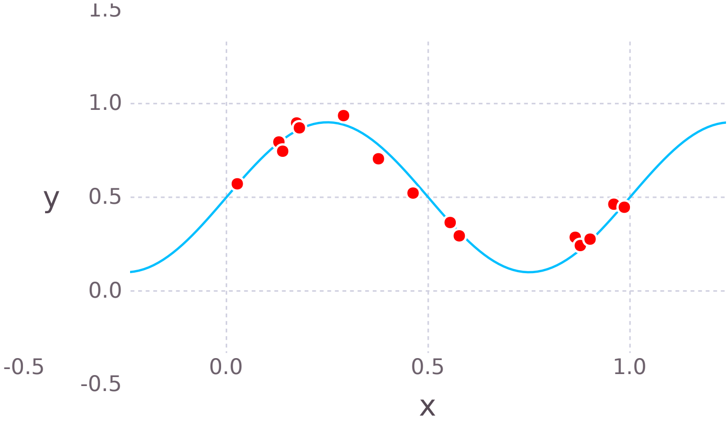


Figure: 15 points from a noisy sinus

Example

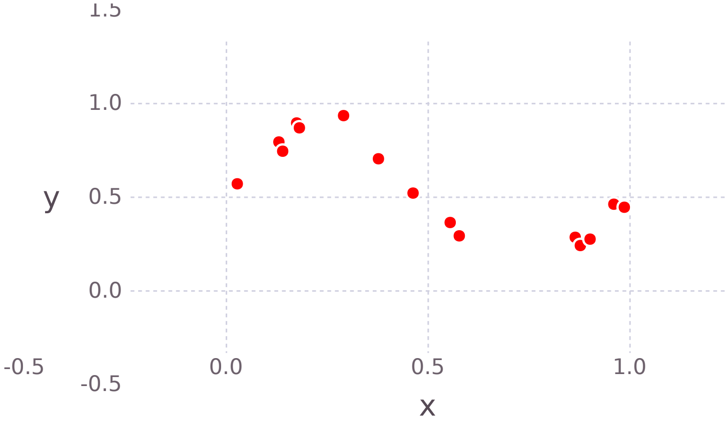


Figure: Exact interpolation using radial basis functions

Example

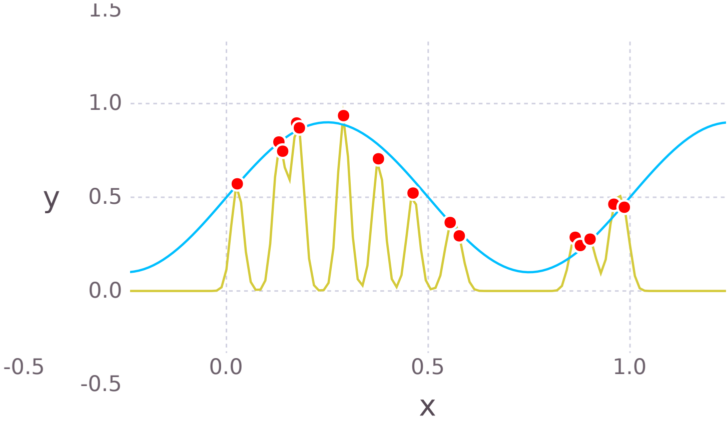


Figure: Kernel: Gaussian, $\sigma = 0.015$

Example

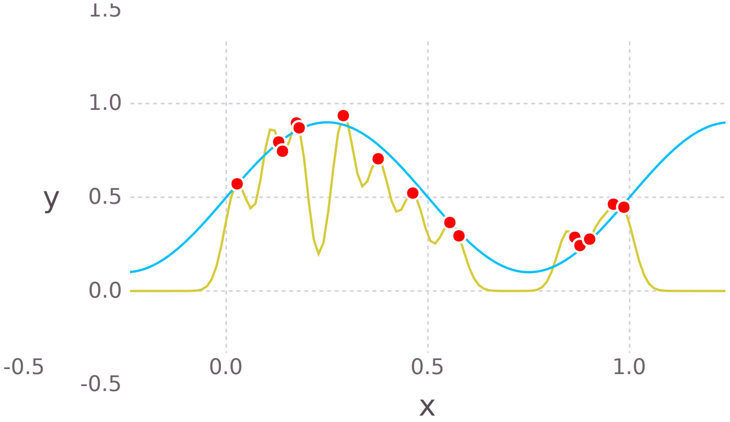


Figure: Kernel: Gaussian, $\sigma = 0.03$

Example

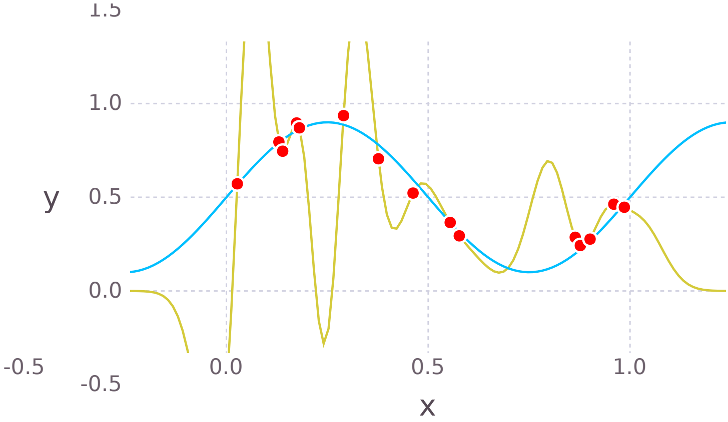


Figure: Kernel: Gaussian, $\sigma = 0.06$

Example

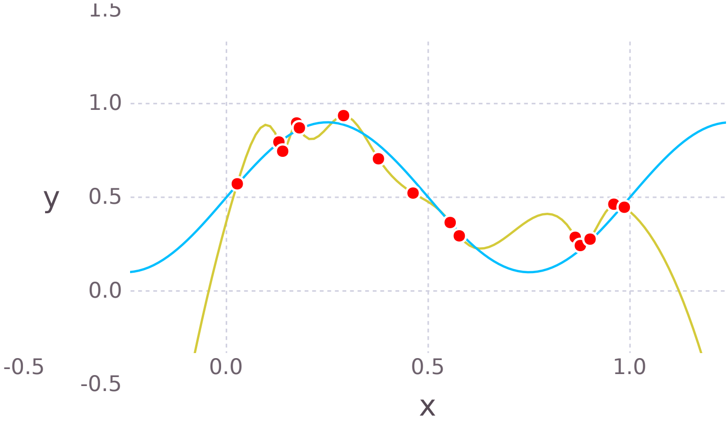


Figure: Kernel: Thin Plate Spline

Example

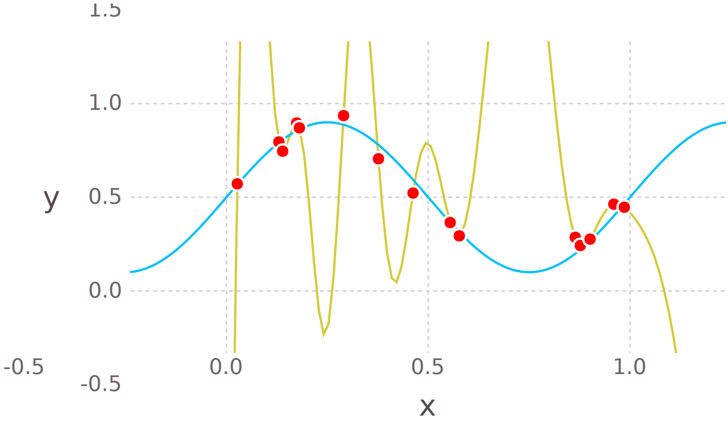


Figure: Kernel: Multi-Quadric $\beta = 0.5$

Example

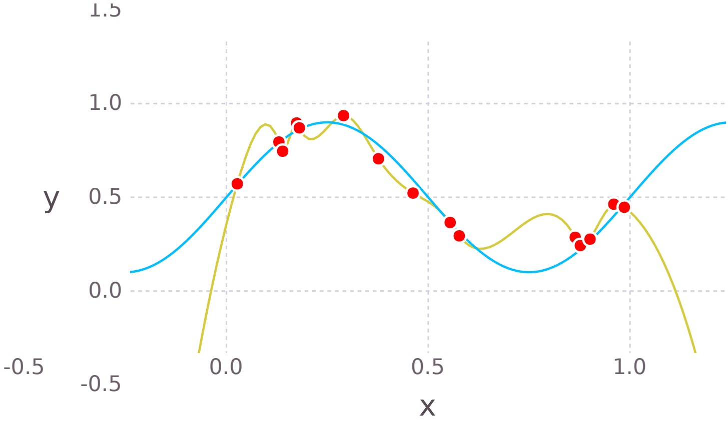


Figure: Kernel: Cubic

Problems

- The function obtained by exact interpolation is a highly oscillatory function.
- If dataset is big, it becomes costly to evaluate.

Today's Outline

Motivation fo RBF Networks

Definition

Cover's theorem: A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.

Motivation fo RBF Networks

Definition

Cover's theorem: A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.

Build a network with two layers such that:

1. the hidden layer applies a transformation form the D-dimensional input space to the M-dimensional feature space (usually $M > D$)
2. the output layer computes a linear combination in the feature space

Separability

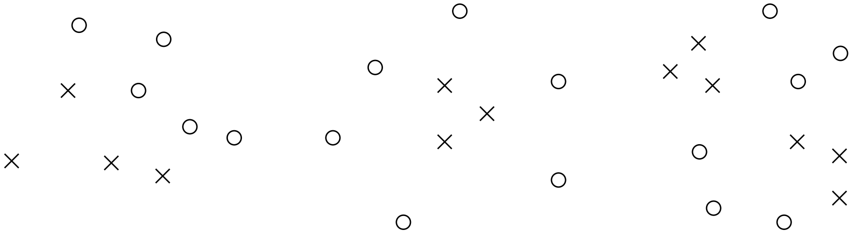


Figure: Dichotomies (adapted from [Hay09])

Separability

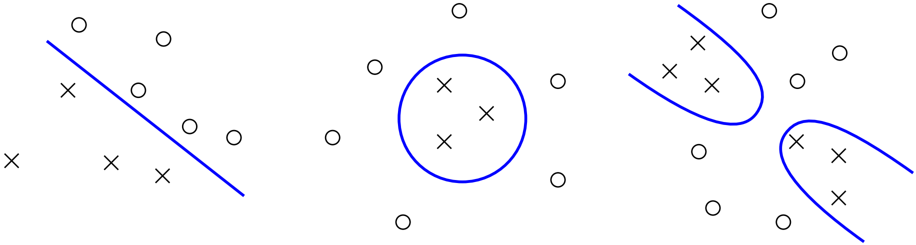


Figure: Dichotomies (adapted from [Hay09])

Radial Basis Function Networks

The interpolation based on radial basis function is modified as follows:

1. much less basis functions than training examples ($M << N$)
2. the centers are not constrained to be examples from the training set
    - finding the prototypes becomes part of the learning process
3. each basis function has its own parameters
    - typically, Gaussians with different variances σ_j
4. biases are added to the linear sum.

Radial Basis Function Networks

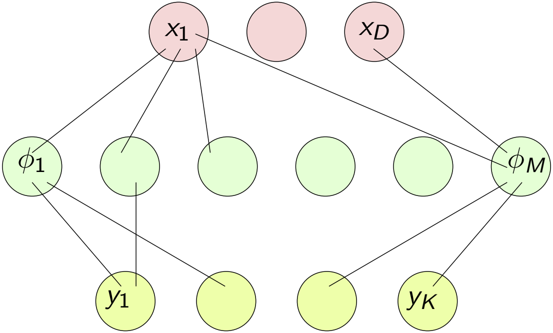
$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0} \tag{11}$$

If basis functions are Gaussians:

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{||\mathbf{x} - \boldsymbol{\mu}_j||^2}{2\sigma_j^2}\right) \tag{12}$$

- $\boldsymbol{\mu}_j$ is the centre of basis function ϕ_j and must be determined during training

Separability



Today's Outline

1. Radial Basis Function Networks
   - The Interpolation Problem
   - RBF Networks
   - RBF Networks for Classification
2. Training Radial Basis Function Networks
3. Comparison with Multi-Layer Feedforward Networks

RBFN for Classification

- Goal in classification: model posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$
- fit each class using a kernel function

$$P(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)P(\mathcal{C}_k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathcal{C}_k)P(\mathcal{C}_k)}{\sum\limits_j p(\mathbf{x}|\mathcal{C}_j)P(\mathcal{C}_j)} \tag{13}$$

Consider a RBFN as follows:

- $\phi_k(\mathbf{x}) = \dfrac{p(\mathbf{x}|\mathcal{C}_k)}{\sum\limits_j p(\mathbf{x}|\mathcal{C}_j)P(\mathcal{C}_j)}$
- $w_{kj} = \begin{cases} p(\mathcal{C}_k) & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases}$

Today's Outline

1. Radial Basis Function Networks
2. Training Radial Basis Function Networks
3. Comparison with Multi-Layer Feedforward Networks

Today's Outline

Network training

- Radial Basis Function Networks are trained using a two-stage hybrid procedure:
  - I Transform the input vectors using basis functions into a higher-dimensional space where is more likely to get linear separability. Determine the parameters of the basis functions using unsupervised learning (only $\mathbf{X}$)
  - II Keep the first-layer parameters fixed and solve the linear problem to determine the second-layer weights.

Today's Outline

The Goal of the First Stage of Training

- The parameters of the basis functions should lead to a representation of the probability density of the training set.
- There are applications where unlabeled data is available, but already classified examples are harder to get.
- **Unsupervised techniques** fit this approach.
- The goal is to learn prototype vectors $\boldsymbol{\mu}_j$.

Learning the prototype vectors

- random subset of the input vectors
    - large number of prototypes needed, poor performance
    - easy to compute, used to initialize iterative algorithms
- iterative selection of basis functions:
    1. choose on input vector and make it a prototype
    2. at step l:
        1. compute $N - l$ networks by adding each input vector left as an additional prototype
        2. compute second-layer weights for all $N - l$ networks
        3. keep the best network from the $N - l$
- *orthogonal least squares*

Clustering Algorithms

- use a clustering algorithm to extract the prototypes of the input vectors
- K-Means
  - fast and efficient
  - optimizes the following cost function:

$$\sum_{j=1}^{K} \sum_{\mathbf{x}^{(n)} \in \mathcal{C}_j} \mathbf{x}^{(n)} \tag{14}$$

K-Means Algorithm

Algorithm 1 K-Means

procedure K-MEANS$(\mathbf{X}, K)$

$\boldsymbol{\mu}_j \leftarrow \mathbf{X}(:, rand() \cdot N), j = 1, \ldots, K$

$\mathcal{C}_j \longleftarrow \{\mathbf{x}^{(n)} |\ ||\mathbf{x}^{(n)} - \boldsymbol{\mu}_j|| \leq ||\mathbf{x}^{(n)} - \boldsymbol{\mu}_i||,\ \forall i = 1, \ldots, K\}$

while $\exists j : \mathcal{C}_j^{\text{old}} \neq \mathcal{C}_j$ **do**

$\mathcal{C}_j^{\text{old}} = \mathcal{C}_j,\ \forall j = 1, \ldots, K$

$\boldsymbol{\mu}_j \longleftarrow \frac{1}{||\mathcal{C}_j||} \sum_{\mathbf{x}^{(n)} \in \mathcal{C}_j} \mathbf{x}^{(n)}$

$\mathcal{C}_j \longleftarrow \{\mathbf{x}^{(n)} |\ ||\mathbf{x}^{(n)} - \boldsymbol{\mu}_j|| \leq ||\mathbf{x}^{(n)} - \boldsymbol{\mu}_i||,\ \forall i = 1, \ldots, K\}$

return $\mathcal{C}_j$

Today's Outline

Optimizing second-layer parameters

The Goal of the Second Stage of Training

For a Radial Basis Function Network, assuming that the parameters of the basis function have already been optimized in the first stage, find a set of weights $\mathbf{W}^*$ that minimize an error function like the sum-of-squares error:

$$E(\mathbf{W}) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} y_k(\mathbf{W}, \mathbf{x}^{(n)}) - t_k^{(n)} \tag{15}$$

where $\mathbf{y}(\mathbf{W}, \mathbf{X}) = \tilde{\mathbf{W}}\tilde{\mathbf{\Phi}}$.

Remember the linear discriminants

- Strategies for finding $\mathbf{W}^{*}$:
  - solving the normal equations using the pseudo-inverse $\mathbf{\Phi}^{\dagger}$:

$$\tilde{\mathbf{\Phi}}^{\top}(\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \mathbf{T}) = \mathbf{0}$$

Remember the linear discriminants

- Strategies for finding $\mathbf{W}^*$:
  - solving the normal equations using the pseudo-inverse $\mathbf{\Phi}^\dagger$:

$$
\begin{aligned}
\tilde{\mathbf{\Phi}}^\top(\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \mathbf{T}) &= \mathbf{0} \\
\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \tilde{\mathbf{\Phi}}^\top\mathbf{T} &= \mathbf{0}
\end{aligned}
$$

Remember the linear discriminants

- Strategies for finding $\mathbf{W}^*$:
  - solving the normal equations using the pseudo-inverse $\mathbf{\Phi}^\dagger$:

$$\begin{aligned}
\tilde{\mathbf{\Phi}}^\top(\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \mathbf{T}) &= \mathbf{0} \\
\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \tilde{\mathbf{\Phi}}^\top\mathbf{T} &= \mathbf{0} \\
\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} &= \tilde{\mathbf{\Phi}}^\top\mathbf{T}
\end{aligned}$$

Remember the linear discriminants

- Strategies for finding $\mathbf{W}^*$:
  - solving the normal equations using the pseudo-inverse $\mathbf{\Phi}^\dagger$:

$$\begin{aligned}
\tilde{\mathbf{\Phi}}^\top(\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \mathbf{T}) &= \mathbf{0} \\
\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \tilde{\mathbf{\Phi}}^\top\mathbf{T} &= \mathbf{0} \\
\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} &= \tilde{\mathbf{\Phi}}^\top\mathbf{T} \\
\tilde{\mathbf{W}} &= (\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}})^{-1}\tilde{\mathbf{\Phi}}^\top\mathbf{T}
\end{aligned}$$

Remember the linear discriminants

- Strategies for finding $\mathbf{W}^*$:
  - solving the normal equations using the pseudo-inverse $\mathbf{\Phi}^\dagger$:

$$
\begin{aligned}
\tilde{\mathbf{\Phi}}^\top(\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \mathbf{T}) &= \mathbf{0} \\
\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \tilde{\mathbf{\Phi}}^\top\mathbf{T} &= \mathbf{0} \\
\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} &= \tilde{\mathbf{\Phi}}^\top\mathbf{T} \\
\tilde{\mathbf{W}} &= (\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}})^{-1}\tilde{\mathbf{\Phi}}^\top\mathbf{T} \\
\tilde{\mathbf{W}} &= \tilde{\mathbf{\Phi}}^\dagger\mathbf{T}
\end{aligned}
$$

Remember the linear discriminants

- Strategies for finding $\mathbf{W}^*$:
  - solving the normal equations using the pseudo-inverse $\mathbf{\Phi}^\dagger$:

$$
\begin{aligned}
\tilde{\mathbf{\Phi}}^\top(\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \mathbf{T}) &= \mathbf{0} \\
\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} - \tilde{\mathbf{\Phi}}^\top\mathbf{T} &= \mathbf{0} \\
\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}}\tilde{\mathbf{W}} &= \tilde{\mathbf{\Phi}}^\top\mathbf{T} \\
\tilde{\mathbf{W}} &= (\tilde{\mathbf{\Phi}}^\top\tilde{\mathbf{\Phi}})^{-1}\tilde{\mathbf{\Phi}}^\top\mathbf{T} \\
\tilde{\mathbf{W}} &= \tilde{\mathbf{\Phi}}^\dagger\mathbf{T}
\end{aligned}
$$

  - singular-value decomposition (SVD) to deal with singular $\mathbf{\Phi}^\top\mathbf{\Phi}$ - the recommended solution
  - recursive algorithm: **LRS** [Hay09]

RLS Algorithm

- RLS Algorithm [Hay09, pag. 245]

Given $\{\phi^{(n)}, \mathbf{t}^{(n)}\}_{1 \leq n \leq N}$, repeat for $n = 1, \ldots, N$:

$$\mathbf{P}^{(n)} = \mathbf{P}^{(n-1)} - \frac{\mathbf{P}^{(n-1)}\phi^{(n)}\phi^{(n)\mathsf{T}}\mathbf{P}^{(n-1)}}{1 + \phi^{(n)\mathsf{T}}\mathbf{P}^{(n-1)}\phi^{(n)}} \tag{16}$$

$$\mathbf{g}^{(n)} = \mathbf{P}^{(n)}\phi^{(n)} \tag{17}$$

$$\alpha^{(n)} = \mathbf{t}^{(n)} - \tilde{\mathbf{W}}^{(n)\mathsf{T}}\phi^{(n)} \tag{18}$$

$$\tilde{\mathbf{W}}^{(n)} = \tilde{\mathbf{W}}^{(n-1)} + \mathbf{g}^{(n)}\alpha^{(n)} \tag{19}$$

where for $n = 0$:

$$\tilde{\mathbf{W}}^{(n)} = \mathbf{0} \tag{20}$$

$$\mathbf{P}(0) = \lambda^{-1}\mathbf{I} \tag{21}$$

Today's Outline

RBFN vs. MLP : Unit activation

RBFN

Activation of hidden units is determined by the distance between the input and a prototype vector.

$$\phi_j = k(||\mathbf{x} - \boldsymbol{\mu}_j||)$$

Activation is constant on $(D-1)$-dimensional hyperplanes.

MLP

Activation of hidden units is determined by the scalar product between the input and a weight vector.

$$z_j = \mathbf{w}_j \mathbf{x} + w_{j0}$$

Activaton is constant on $(D-1)$-dimensional hyperspheres around the prototype vector.

RBFN vs. MLP : Training

RBFN

Faster training for the RBFN.

Two stage training procedure:

1. set the parameters for the basis functions
2. optimize the weights for the last layer
   - linear problem

MLP

Slower training.

All weights are updated at once.

RBFN vs. MLP : Training

RBFN

Only a few hidden units are activated by an input: **representation** in the space of hidden units is local with respect to the input space.

MLP

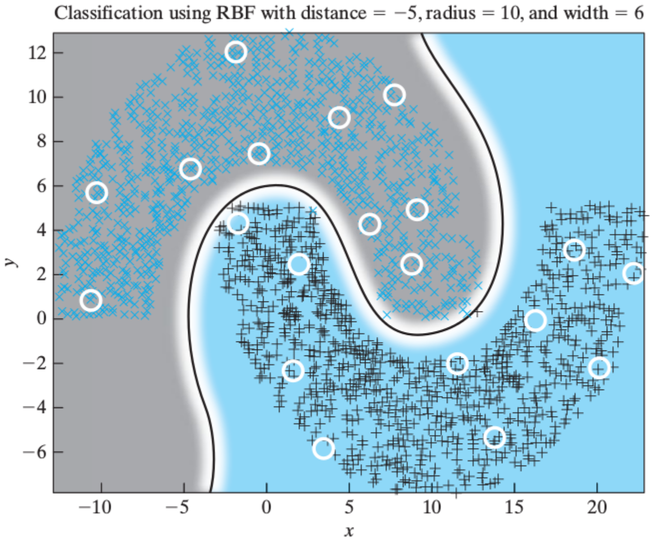
A complex interplay of hidden units yields the output result. MLPs have a distributed representation in the hidden units space.

RBFN vs. MLP : Training

RBFN	MLP
Usually only two layers.	Can be scaled to many layers, with various patterns of connectivity and different activation functions.

Laboratory Work

Implement an RBF network to solve the two-moon problem.



Summary

- RBFNs have their roots in the interpolation theory
- RBFNs are trained in a two-stage procedure:
  - unsupervised learning for radial basis functions' parameters
  - linear optimization for the output layer's weights

For the exam

For the exam you should be able to ...

- describe the radial-basis functions solution to the exact interpolation problem
- describe how a radial-basis function network works
- describe the two-stage training procedure for the RBFNs

Read ...

- Chapter 5, *Kernel Methods and Radial-Basis Function Networks* from [Hay09]

Today's Outline

References I

- Simon S. Haykin, *Neural networks and learning machines*, Prentice Hall, 2009.
- M. J. D. Powell, *Algorithms for approximation*, Clarendon Press, New York, NY, USA, 1987, pp. 143–167.