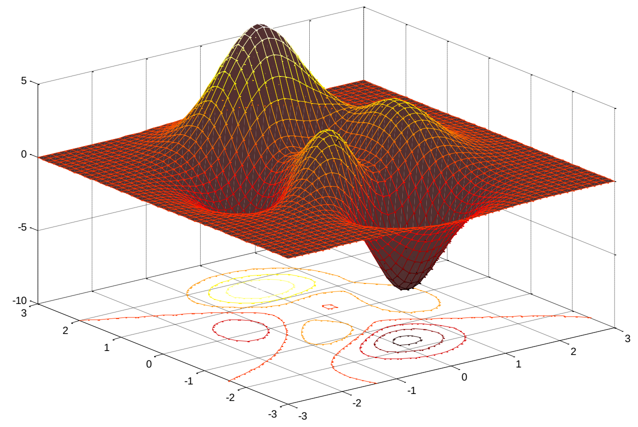
IN THE NAME OF ALLAH

Neural Networks

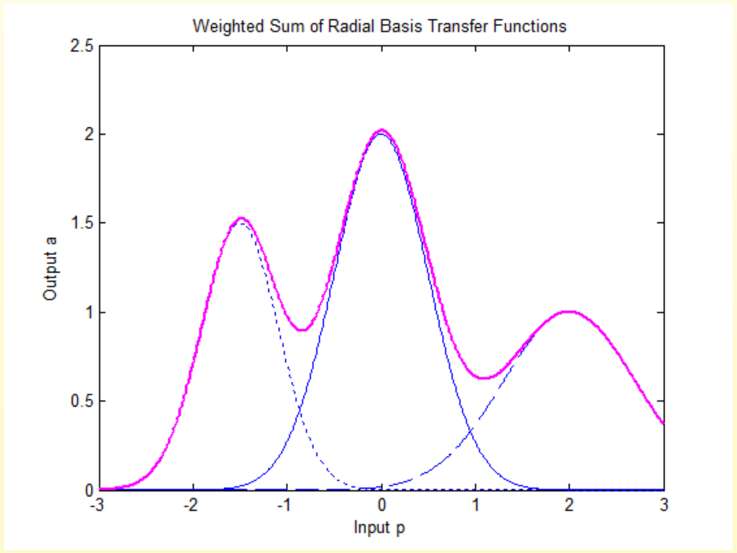
Radial Basis Function Networks

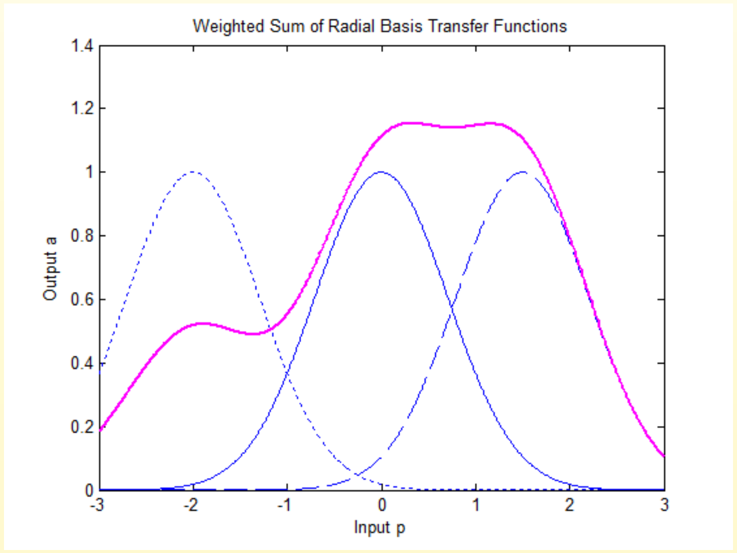


Shahrood University of Technology
Hossein Khosravi

RBF vs. FFT!

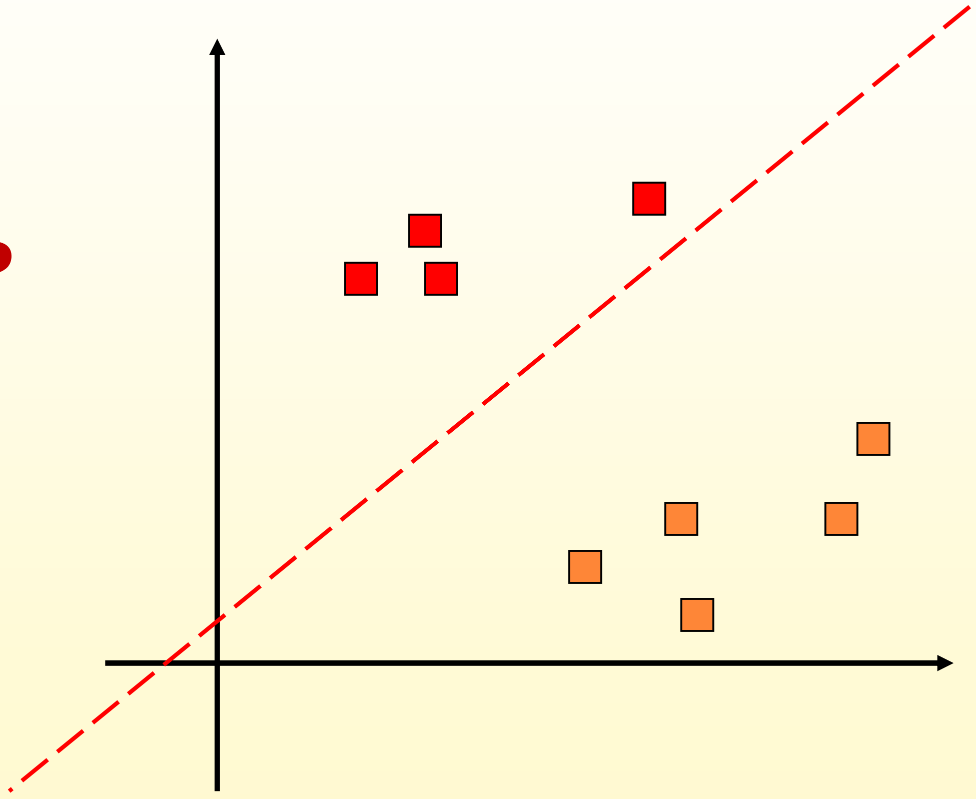
- A function is radial basis (**RBF**) if its output depends on the distance of the input from a given stored vector (a non-increasing function).
- FFT Idea: Almost any signal is a combination of sinusoids of different frequencies and amplitudes.
- RBF Idea: Almost any function can be approximated using mixture of Gaussians with different sigma and centers.





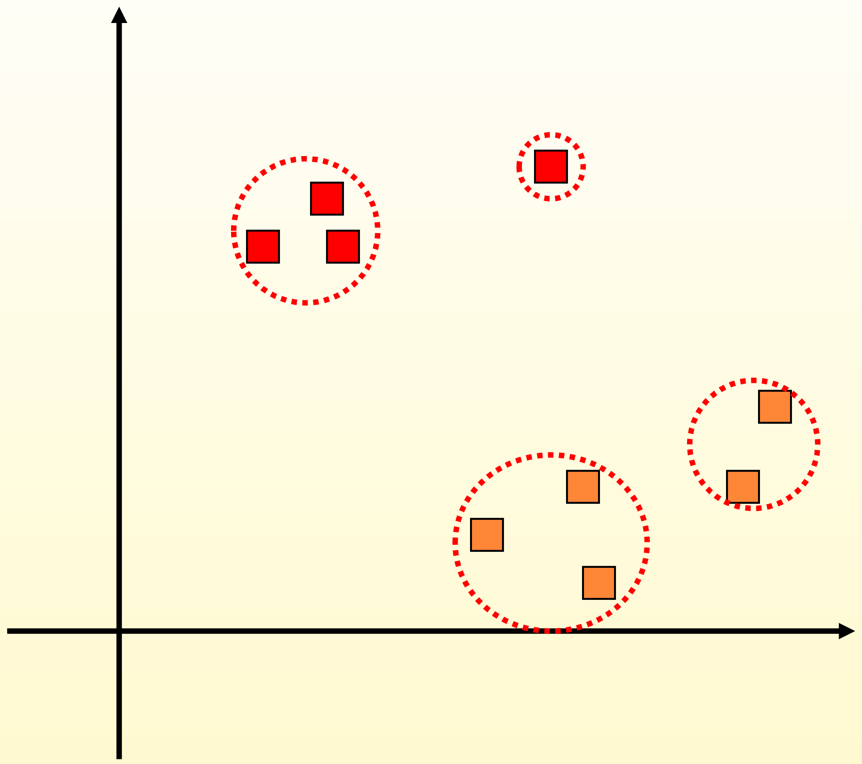
RBF vs. MLP

In MLP

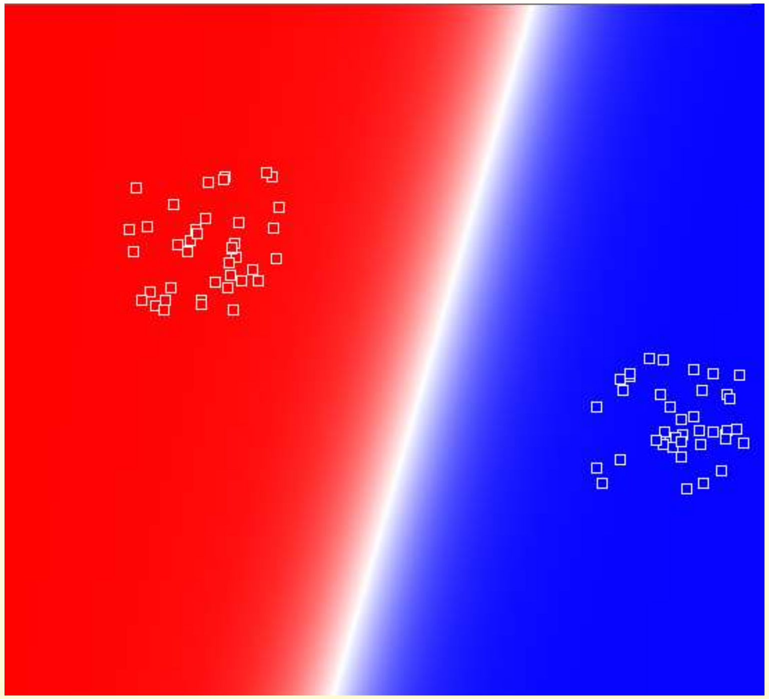


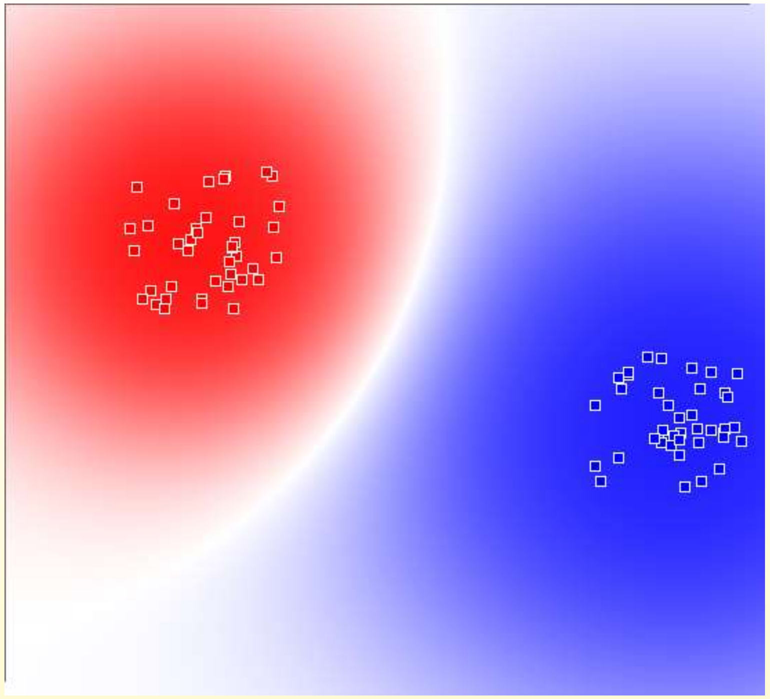
RBF vs. MLP

In RBFN



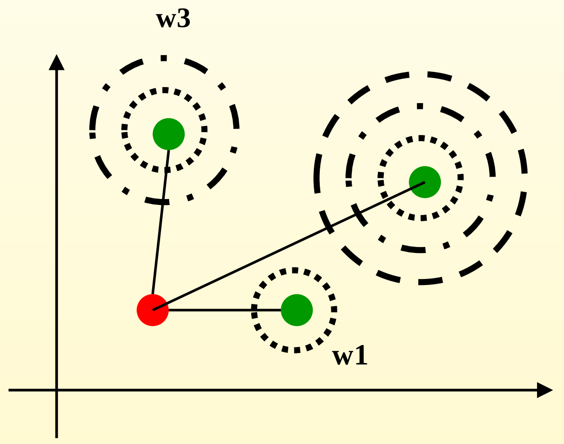
RBF vs. MLP





Radial Basis Function Network

- 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

$$w1 < w3 < w2$$

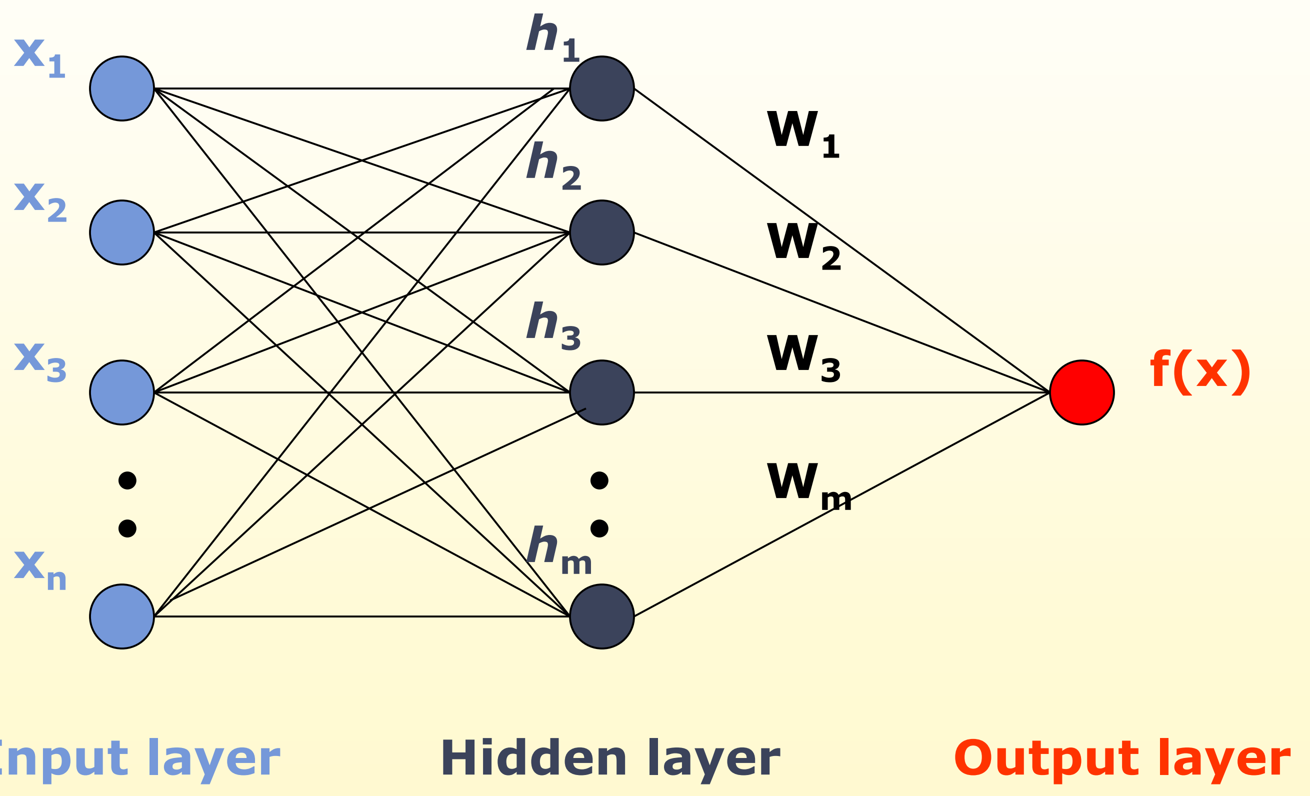
Radial Basis Function Network

- Approximate function with linear combination of Radial basis functions

$$F(x) = \Sigma \; w_i \, h(x)$$

- $h(x)$ is mostly Gaussian function

Architecture



Three layers

- 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

Radial basis function

- Design Requires
  - Selection of the RBF width parameter (σ)
  - Number of radial basis neurons

$$f(X) = \sum_{j=1}^{n} W_j h_j(X)$$

$$h_j(X) = \exp\left(-\frac{\left\|X - \mu_j\right\|^2}{2\sigma_j^2}\right)$$

j is index of the hidden neuron,
X is the input vector,
μ_j is mean vector or prototype vector of *j*th neuron
and σ_j is the spread parameter

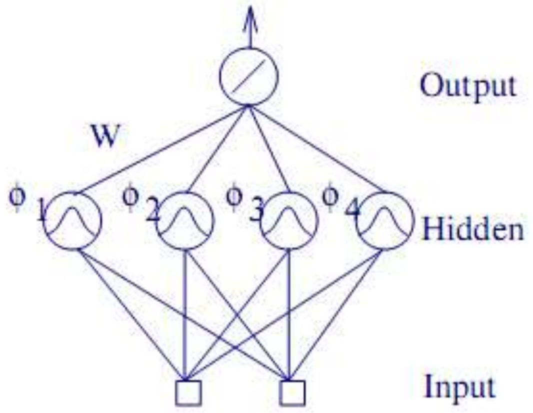
RBF Learning: Overview

- The RBF learning problem boils down to two tasks:

  - How to determine the parameters associated with the radial-basis functions in the hidden layer $\varphi_i(x)$ (e.g., the center of the Gaussians).

  - How to train the hidden-to-output weights?
    This part is relatively easy.

RBF as an Interpolation Problem



- m_0-D input to 1-D output mapping $s : \mathbb{R}^{m_0} \longrightarrow \mathbb{R}^1$.
- The map s can be thought of as a *hypersurface* $\Gamma \subset \mathbb{R}^{m_0+1}$.
  - **Training**: fit hypersurface Γ to the training data points.
  - **Generalization**: interpolate between data points, along the reconstructed surface Γ.
- Given $\{\mathbf{x}_i \in \mathbb{R}^{m_0} | i = 1, 2, \ldots, N\}$ and N labels $\{d_i \in \mathbb{R}^1 | i = 1, 2, \ldots, N\}$, find $F : \mathbb{R}^N \longrightarrow \mathbb{R}^1$ such that

$$F(\mathbf{x}_i) = d_i \text{ forall } i.$$

RBF and Interpolation

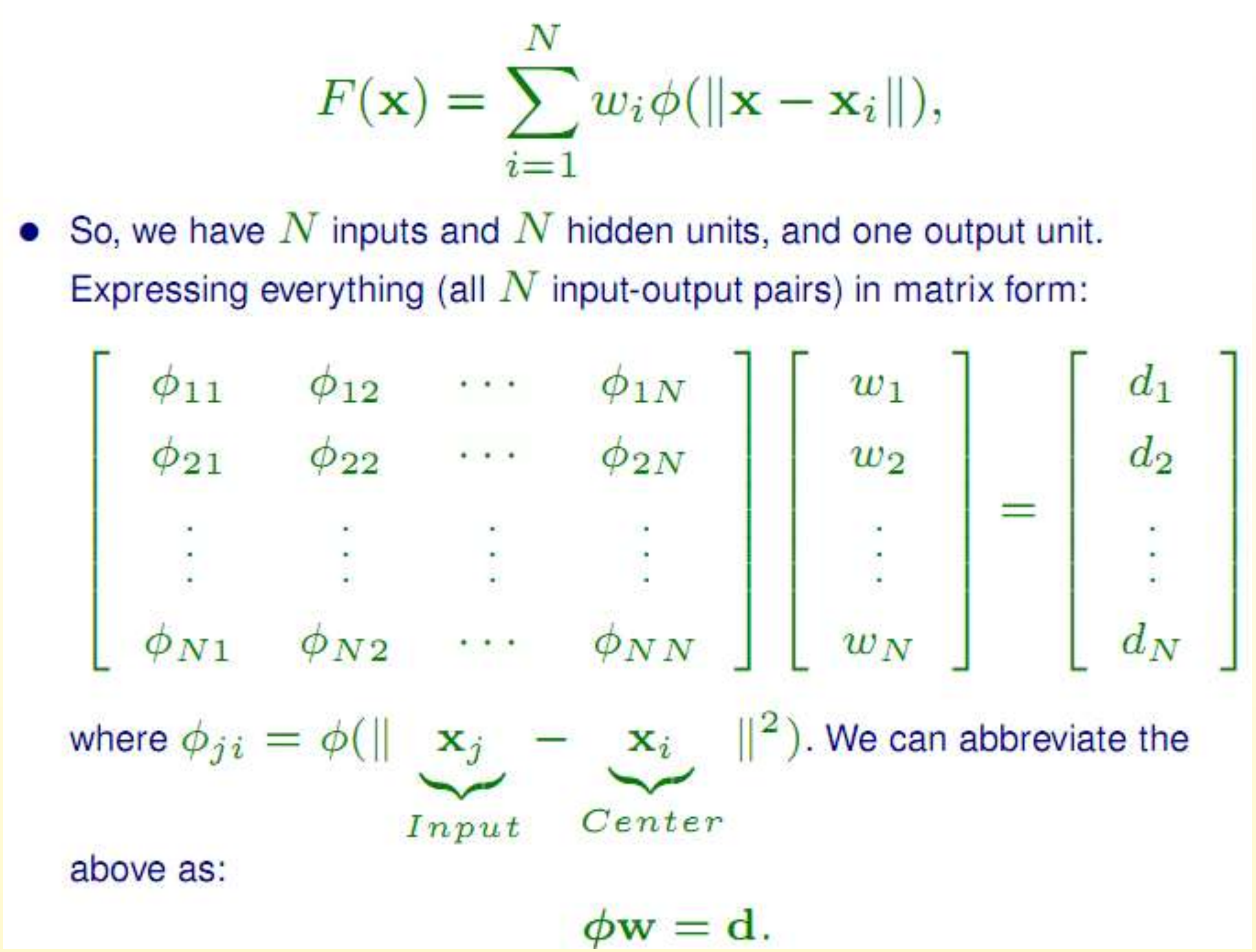
- Interpolation is formulated as

$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(\|\mathbf{x} - \mathbf{x}_i\|),$$

where $\{\phi(\|\mathbf{x} - \mathbf{x}_i\|) | i = 1, 2, \ldots, N\}$ is a set of N arbitrary (nonlinear) functions known as *radial-basis functions*.

- The *known data points* $\mathbf{x}_i \in \mathbb{R}^{m_0}, i = 1, 2, \ldots, N$ are treated as the *centers* of the RBFs. (Note that in this case, all input data need to be memorized, as in instance-based learning, but this is not a necessary requirement.)

RBF and Interpolation (cont'd)



$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(\|\mathbf{x} - \mathbf{x}_i\|),$$

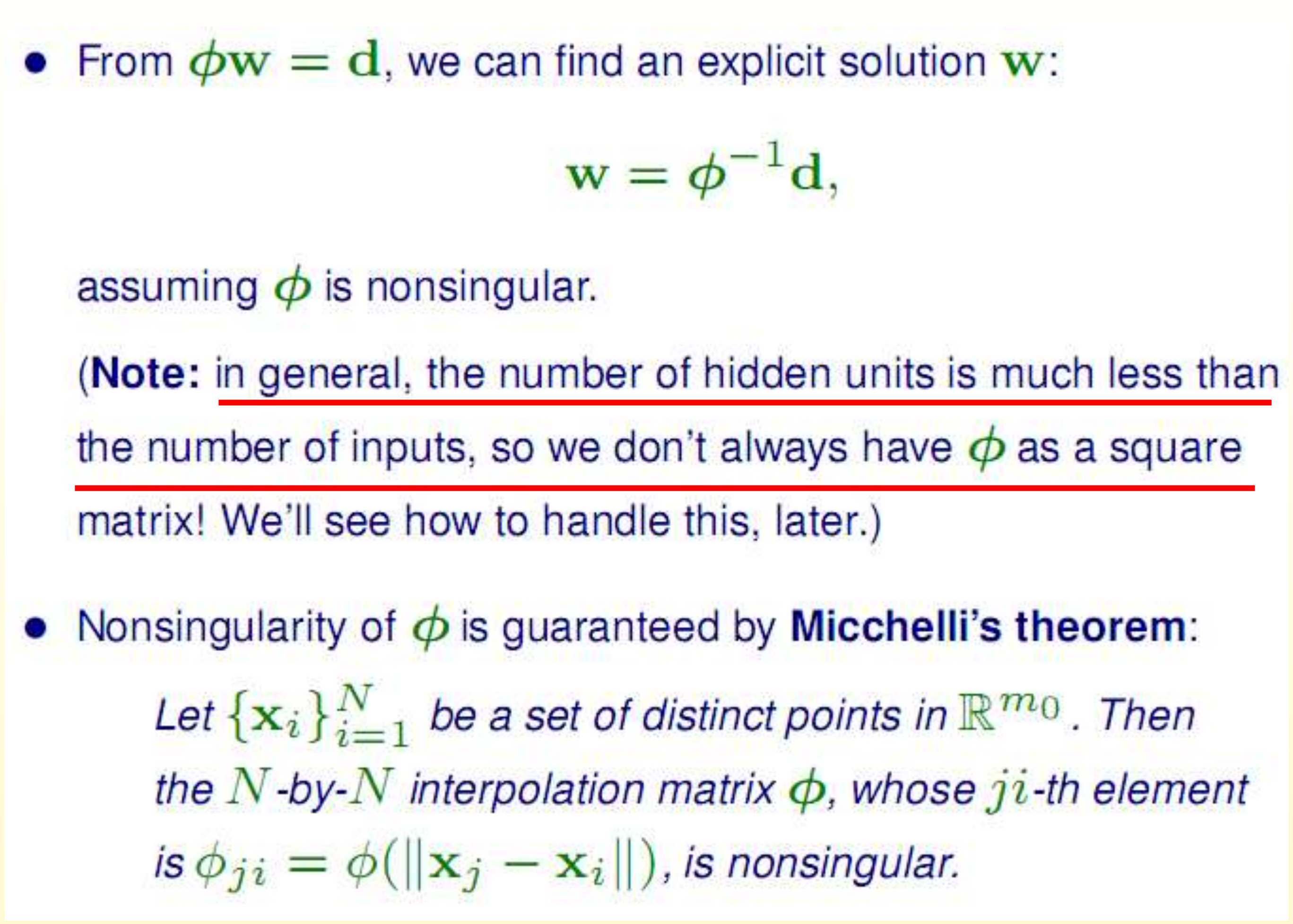
- So, we have N inputs and N hidden units, and one output unit. Expressing everything (all N input-output pairs) in matrix form:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix}$$

where $\phi_{ji} = \phi(\| \underbrace{\mathbf{x}_j}_{Input} - \underbrace{\mathbf{x}_i}_{Center} \|^2)$. We can abbreviate the above as:

$$\phi \mathbf{w} = \mathbf{d}.$$

RBF and Interpolation (cont'd)



- From $\phi\mathbf{w} = \mathbf{d}$, we can find an explicit solution $\mathbf{w}$:

$$\mathbf{w} = \phi^{-1}\mathbf{d},$$

assuming ϕ is nonsingular.

(**Note:** in general, the number of hidden units is much less than the number of inputs, so we don't always have ϕ as a square matrix! We'll see how to handle this, later.)

- Nonsingularity of ϕ is guaranteed by **Micchelli's theorem**:

  Let $\{\mathbf{x}_i\}_{i=1}^N$ be a set of distinct points in $\mathbb{R}^{m_0}$. Then the N-by-N interpolation matrix ϕ, whose ji-th element is $\phi_{ji} = \phi(||\mathbf{x}_j - \mathbf{x}_i||)$, is nonsingular.

RBF and Interpolation (cont'd)

- When $m_0 < N$ (m_0: number of hidden units; N: number of inputs), we can find w that minimizes

$$\mathcal{E}(\mathbf{w}) = \sum_{i=1}^{N} (F(\mathbf{x}_i) - d_i)^2 \,,$$

where $F(\mathbf{x}) = \sum_{k=1}^{m_0} w_k \phi_k(\mathbf{x})$.

- The solution involves the pseudo inverse of ϕ:

$$\mathbf{w} = \underbrace{\left(\phi^T \phi\right)^{-1} \phi^T}_{\text{pseudo inverse}} \mathbf{d}.$$

Note: ϕ is an $N \times m_0$ rectangular matrix.

- In this case, how to determine the centers of the $\phi_k(\cdot)$ functions becomes an issue.

Selection of Spread Parameter

The width σ of Gaussian functions can be fixed according to the spread of the centres:

$$\sigma = \frac{d_{\max}}{\sqrt{2m_1}} \qquad (8.6)$$

where m_1 is the number of centres and d_{max} is the maximum distance between the chosen centres. This formula ensures that the individual radial-basis functions are not too peaked or too flat; both of these extreme conditions should be avoided.

The width σ_k of a Gaussian function $G(x,c_k)$ can also be calculated by so called *P-nearest neighbour heuristic*. Consider a given centre c_k and assume that $c_1, c_2, \ldots, c_P$ are P nearest centres. Then we set

$$\sigma_k = \sqrt{\frac{1}{P}\sum_{i=1}^{P}\left\|c_k - c_i\right\|^2} \qquad (8.7)$$

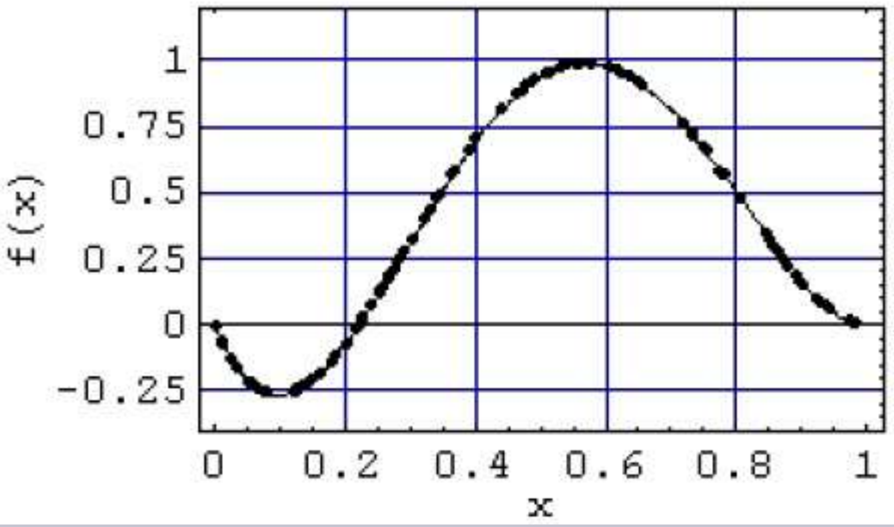
Selection of Centers

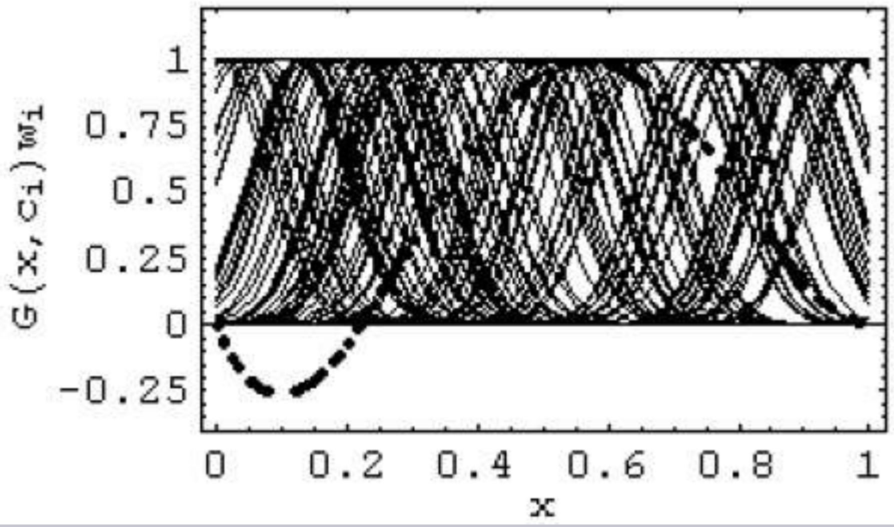
- Random from input space
- Unsupervised (Clustering)
- Supervised (Through Learning)

- Will discussed later

Example of function approximation – large RBFN

- The training data consists of 100 points.
- Therefore, m1 = 100 hidden layer neurons
- Centered at training samples.
- All have the same width calculated from: $\sigma = \frac{d_{max}}{\sqrt{2m_1}}$



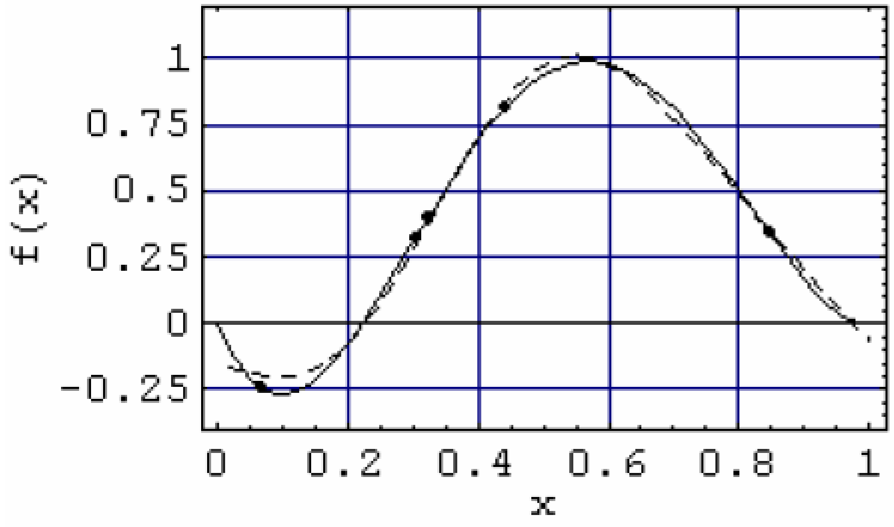


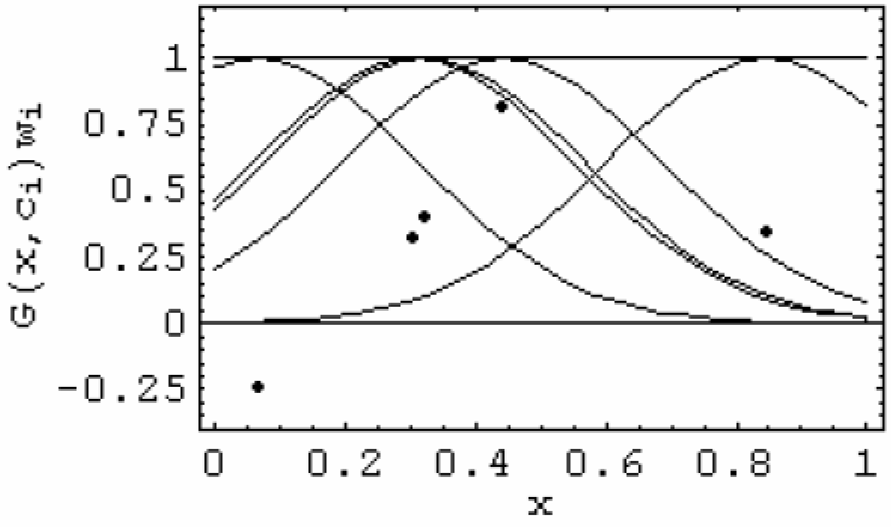
RBFN output (dashed line)

Ensemble of 100 elementary functions

Example of function approximation – small RBFN

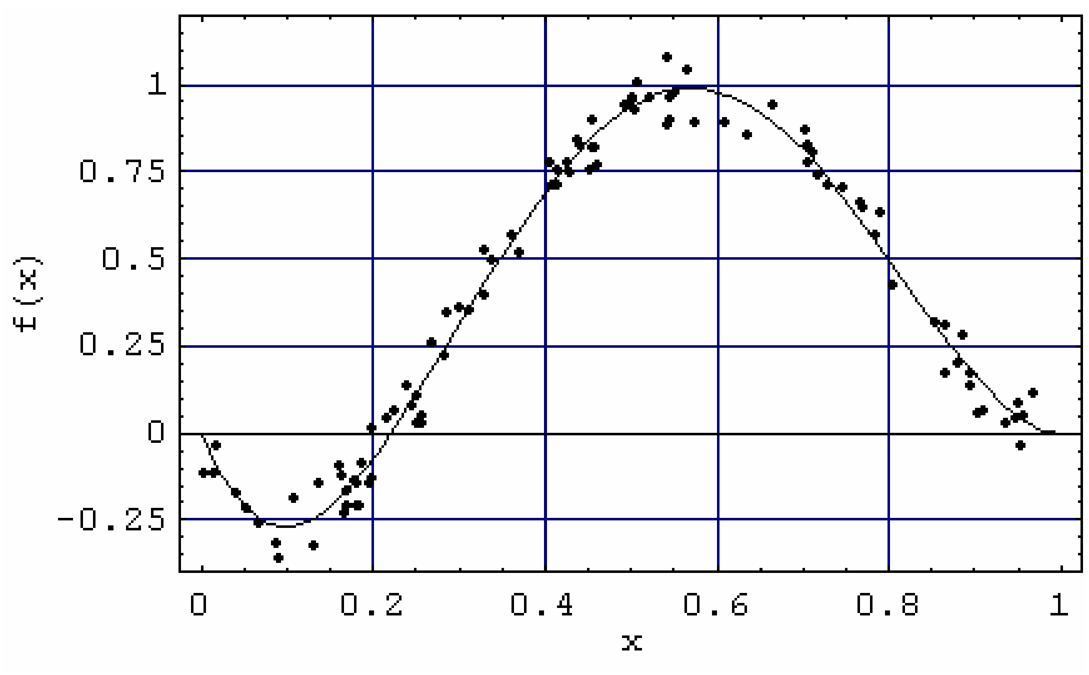
- Training data selected randomly from the training set consist of 5 points.
- RBFN has, therefore, m1 = 5 hidden neurons
- Centered at selected training samples.
- All have the same width calculated as before.



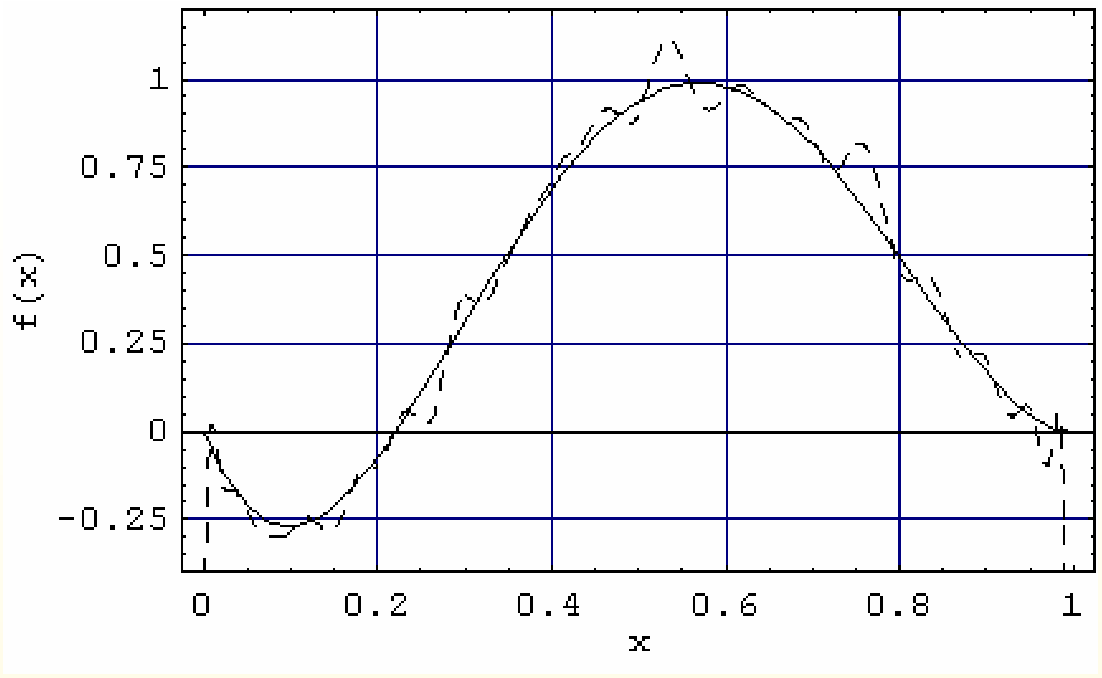


Example of function approximation – noisy data

- Uniformly distributed noise from the interval [-0.1, 0.1] was added to the training samples

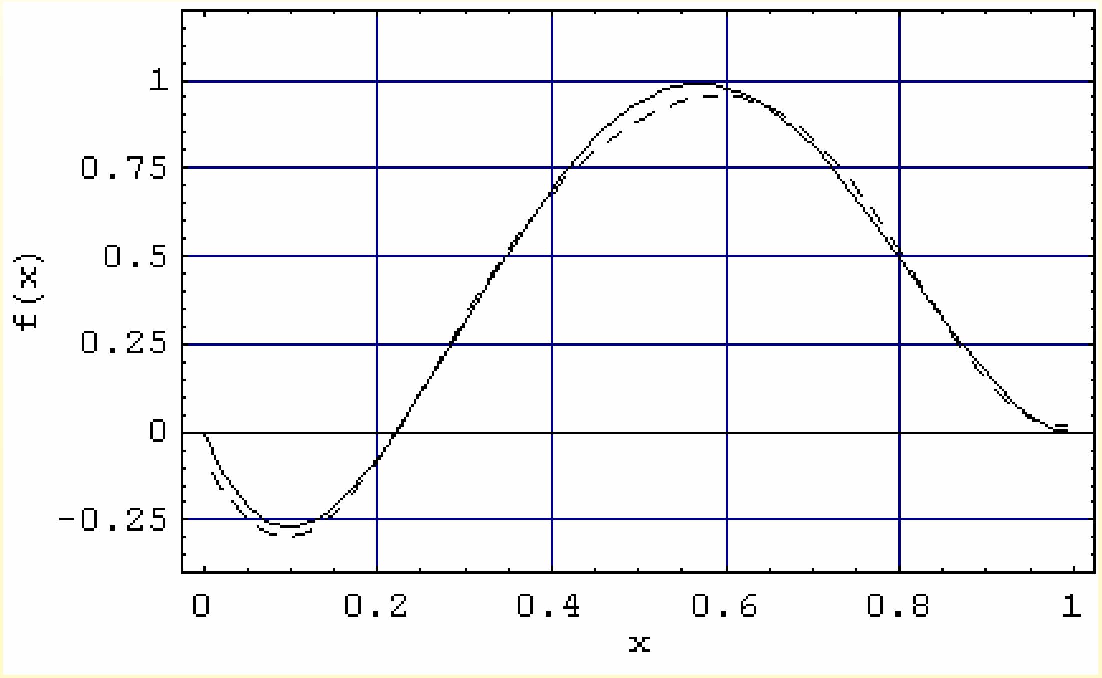


m1 = 100



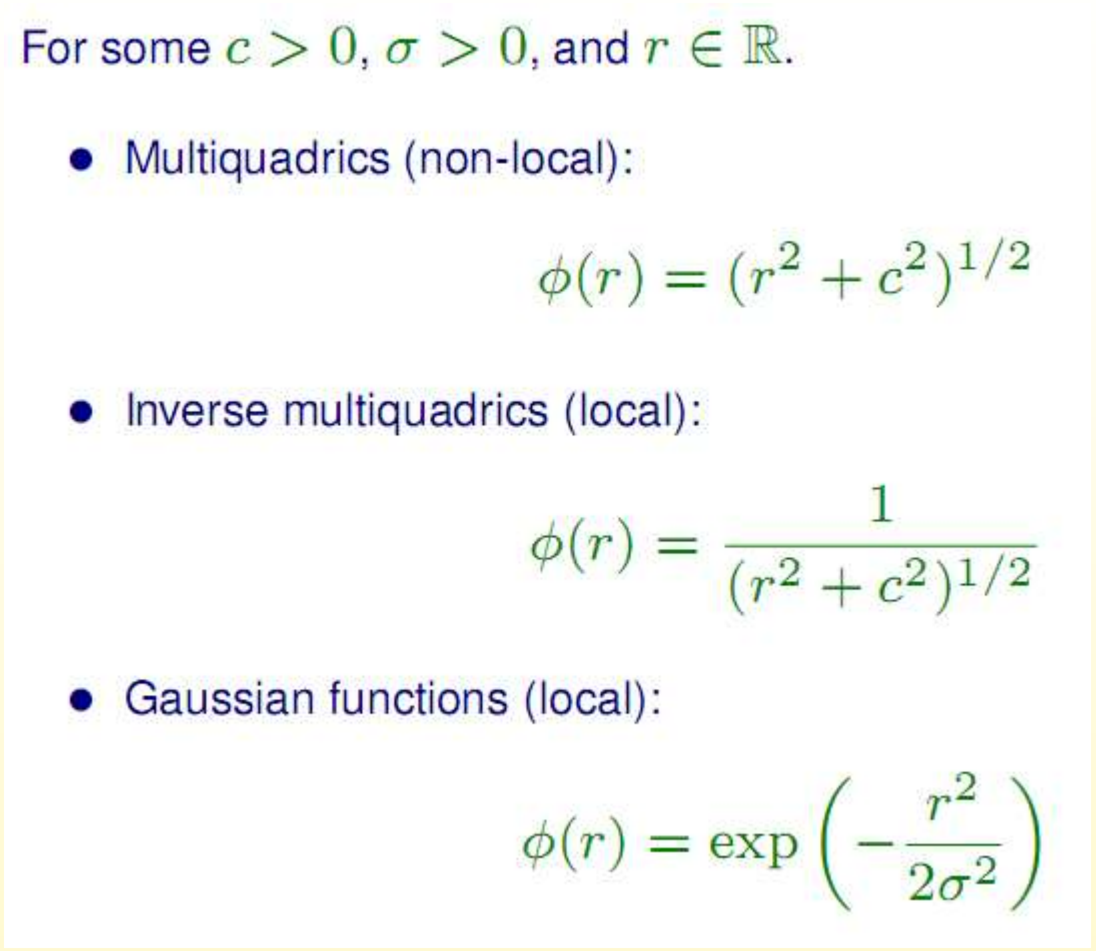
1
0.75
0.5
0.25
0
-0.25
f(x)
0
0.2
0.4
0.6
0.8
1
x

m1 = 5



1
0.75
0.5
0.25
0
-0.25
f(x)
0
0.2
0.4
0.6
0.8
1
x

Typical RBFs



For some $c > 0$, $\sigma > 0$, and $r \in \mathbb{R}$.

- Multiquadrics (non-local):

$$\phi(r) = (r^2 + c^2)^{1/2}$$

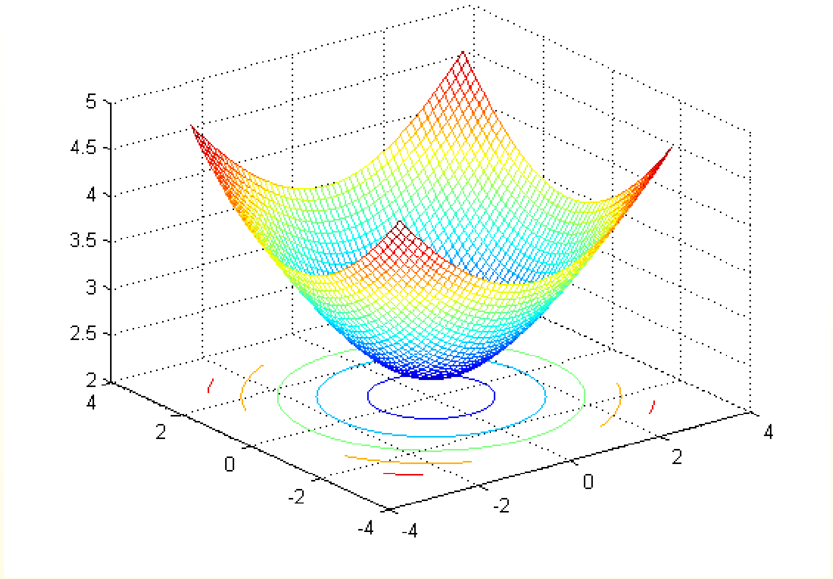
- Inverse multiquadrics (local):

$$\phi(r) = \frac{1}{(r^2 + c^2)^{1/2}}$$

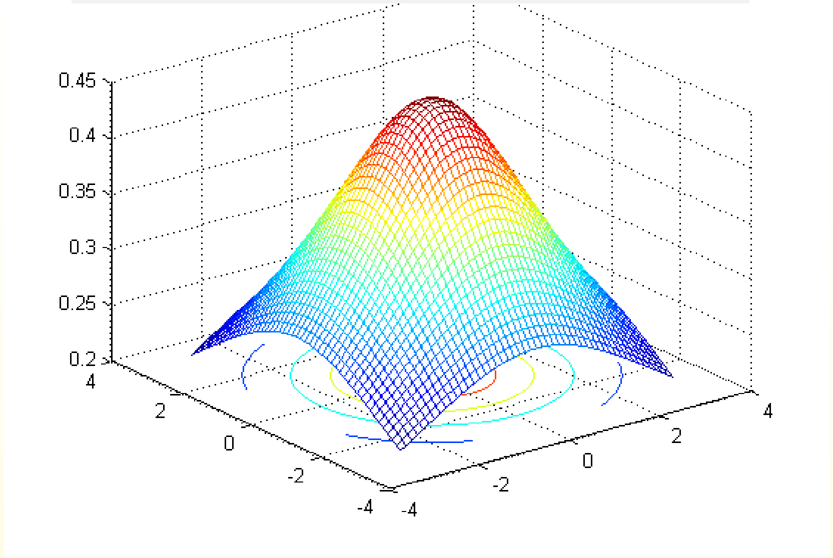
- Gaussian functions (local):

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

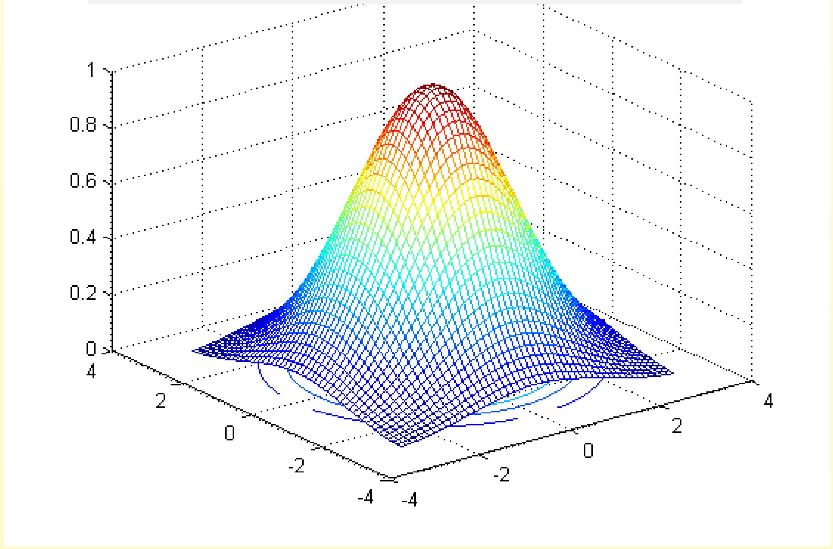
```
Z = sqrt((X.*X+Y.*Y+5));
```



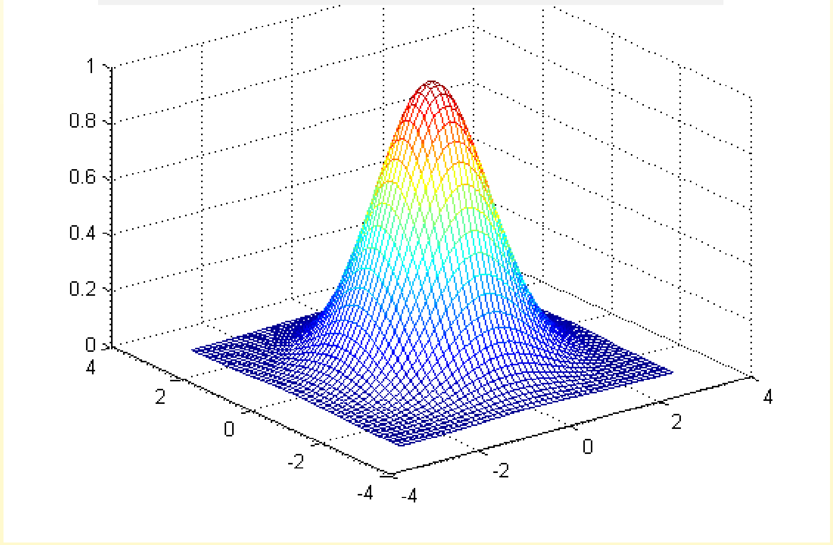
```
[X,Y] = meshgrid(-3:.125:3);
Z = 1./sqrt((X.*X+Y.*Y+5));
meshc(X,Y,Z);
```



```
Z = exp(-0.25*(X.*X+Y.*Y));
```



```
Z = exp(-0.5*(X.*X+Y.*Y));
```



RBF for classification problems

- Number of radial basis neurons
  - By designer
  - Max of neurons = number of input samples
  - Min of neurons = experimentally determined
  - More neurons
    - More complex, but smaller tolerance

Selection of the RBF width para.

- Not required for an MLP
- Small width
  - Cause in untrained data
  - More hidden neuron required
- Large width
  - Network of smaller size & faster execution
- Adaptive width
  - Through training

learning strategies

- Two levels of Learning
  - Center and spread learning (or determination)
  - Output layer Weights Learning

- Make number of parameters as small as possible

Various learning strategies

- how the centers of the radial-basis functions of the network are specified:
  - Fixed centers selected at random
  - Self-organized selection of centers
  - Supervised selection of centers

Fixed centers selected at random

- Fixed RBFs of the hidden units

- The locations of the centers may be chosen randomly from the training data set.

- We can use different values of centers and widths for each radial basis function ➔ Experimentation with training data is needed.

Fixed centers selected at random(cnt'd)

- Only output layer weights must be learned.
- Obtain the value of the output layer weight by pseudo-inverse method (as mentioned before)
- Main problem
  - May not present great performance as MLP
  - Generalization is not great
  - Computation of Inverse Matrix in the case of large hidden size

Self-organized selection of centers(1)

- Hybrid learning
  - self-organized learning to estimate the centers of RBFs in hidden layer
  - supervised learning to estimate the linear weights of the output layer
- Self-organized learning of centers by means of clustering.
- Supervised learning of output weights by LMS algorithm.

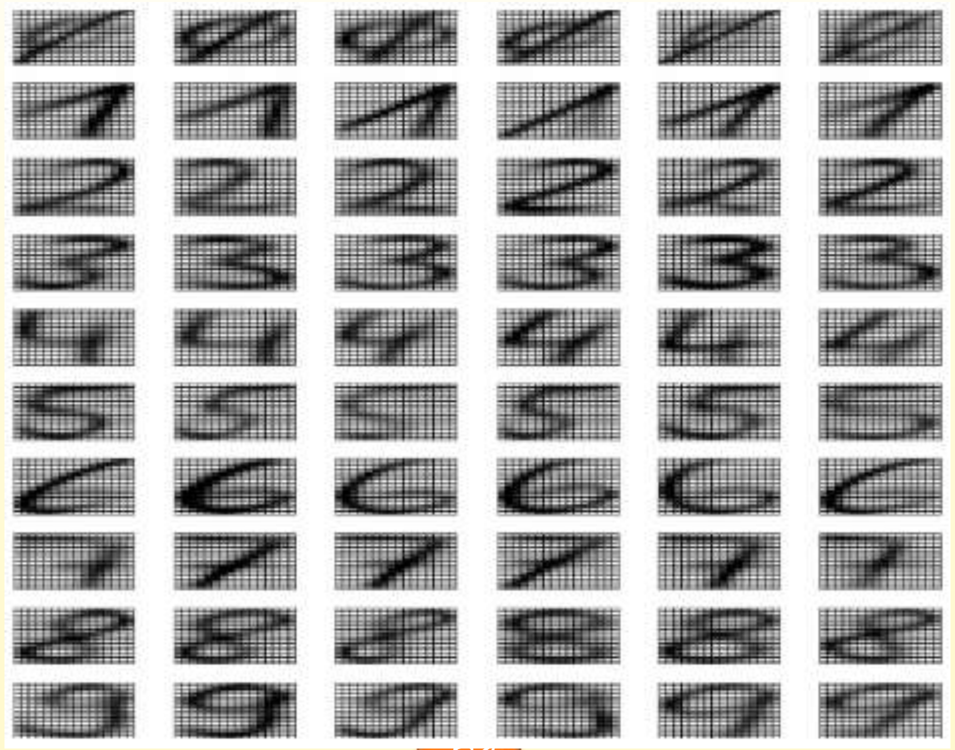
Self-organized selection of centers(2)

- k-means clustering
  1. Initialization
  2. Sampling
  3. Similarity matching
  4. Updating
  5. Continuation

K-Means Animation

Example

- 60 hidden units
  - K-means centers



Supervised selection of centers

- All free parameters of the network are changed by supervised learning process.
- Error-correction learning using LMS algorithm.

RBF Learning

No Training

- **Usually in the case of function approximation, as mentioned before. Weights are computed through matrix inverse.**
- **Try newrbe in MATLAB ("e" stands for Exact)**

Half Training

- **Hidden layer parameters are fixed. Output layer weights are trained.**
- **newrb in MATLAB**

Full Training

- **All parameters are determined through training. No built-in function in MATLAB, I think!**
- **We discuss full training in detail.**

RBF Full Learning - Assumptions

- Hidden neurons are Gaussian:
- Output neurons could be sigmoid, linear, or pseudo-linear, i.e. linear with some squashing property:

$$y_m = f_m(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{v}_\mathrm{m}\|^2}{2\sigma_\mathrm{m}^2}\right)$$

$$z_j = \begin{cases} \dfrac{1}{1+\mathrm{e}^{-S_j}}, & \text{sigmoid,} \\ \dfrac{s_j}{l_2}, & \text{linear, with } \dfrac{1}{l_2} \text{ squashing function,} \\ \dfrac{s_j}{\sum_m y_m}, & \text{pseudo-linear, with } \dfrac{1}{\sum_m y_m} \text{ squashing function,} \end{cases}$$

where

$$s_j = \sum_{m=1}^{l_2} y_m u_{mj}, \quad j = 1, \ldots, l_3.$$

RBF Learning - Initialization of Centers

The first samples of the training set

Some randomly chosen samples from the training set

Centers obtained by some clustering or classification method, e.g. k-means algorithm or LVQ algorithm.

RBF Learning - Initialization of spread parameter

- Assigning a small fixed value, say, 0.05 or 0.1
  - requires a large number of hidden neurons to cover the input space.

- $\sigma = \frac{d}{\sqrt{2l_2}}$
  - where d is the maximum distance between the chosen centers, and l_2 is the number of centers.

- When using k-means to find the kernel vectors, σ_m could be the standard deviation of the vectors in the pertaining cluster.

RBF Learning - Initialization of output layer weights

- Some random values in the range [− 0.1, +0.1].
  - This method necessitates weight adjustment through an iterative process (the backpropagation algorithm).

- Using the pseudo-inverse matrix:

$$\mathbf{w} = \underbrace{\left(\phi^T \phi\right)^{-1} \phi^T}_{\text{pseudo inverse}} \mathbf{d}.$$

Basic backpropagation for the RBF network

- 1. Initialize Network
- 2. Forward pass: Insert the input and the desired output, compute the network outputs
- 3. Backward pass: Calculate the error gradients versus the parameters

$\partial E/\partial u_{mj},\ \partial E/\partial v_{im},\ \partial E/\partial \sigma_m^2$

- 4. Update parameters

$$u_{mj}(n+1) = u_{mj}(n) - \eta_3 \frac{\partial E}{\partial u_{mj}},$$

$$v_{im}(n+1) = v_{im}(n) - \eta_2 \frac{\partial E}{\partial v_{im}},$$

$$\sigma_m^2(n+1) = \sigma_m^2(n) - \eta_1 \frac{\partial E}{\partial \sigma_m^2},$$

- 5. Repeat the algorithm for all training inputs in several epochs.

Updating Parameters

$$u_{mj}(n+1) = u_{mj}(n) - \eta_3 \frac{\partial E}{\partial u_{mj}},$$

$$v_{im}(n+1) = v_{im}(n) - \eta_2 \frac{\partial E}{\partial v_{im}},$$

$$\sigma_m^2(n+1) = \sigma_m^2(n) - \eta_1 \frac{\partial E}{\partial \sigma_m^2},$$

$$\frac{\partial E}{\partial u_{mj}} = \overbrace{\frac{\partial E}{\partial z_j}}^{\text{I}} \overbrace{\frac{\partial z_j}{\partial s_j}}^{\text{II}} \overbrace{\frac{\partial s_j}{\partial u_{mj}}}^{\text{III}}$$

$$\boxed{-2(t_j - z_j) z_j \, (1 - z_j) y_m}$$

$$\frac{\partial E}{\partial v_{im}} = \sum_j \overbrace{\frac{\partial E}{\partial z_j}}^{\text{I}} \overbrace{\frac{\partial z_j}{\partial y_m}}^{\text{II}} \overbrace{\frac{\partial y_m}{\partial v_{im}}}^{\text{III}}. \rightarrow \boxed{\sum_j -2(t_j - z_j) z_j (1 - z_j) u_{mj} \frac{y_m}{\sigma_m^2} (x_{im} - v_{im});}$$

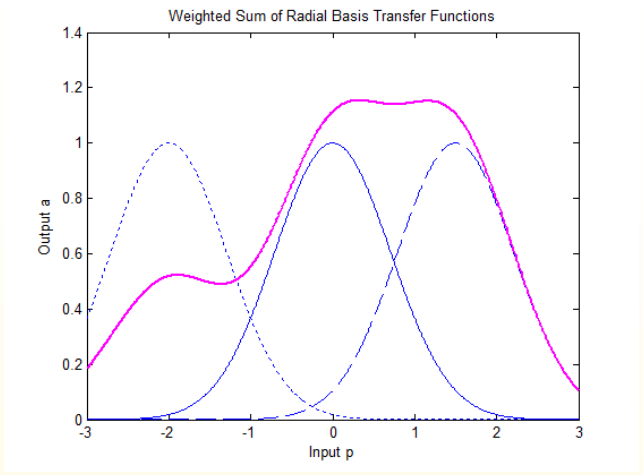
$$\frac{\partial E}{\partial \sigma_m^2} = \sum_j \overbrace{\frac{\partial E}{\partial z_j}}^{\text{I}} \overbrace{\frac{\partial z_j}{\partial y_m}}^{\text{II}} \overbrace{\frac{\partial y_m}{\partial \sigma_m^2}}^{\text{III}}, \rightarrow \boxed{\sum_j -2(t_j - z_j) z_j (1 - z_j) u_{mj} y_m \left(\frac{\|\mathbf{x} - \mathbf{v}_m\|^2}{2\sigma_m^4} \right)}$$

Gradient Computation

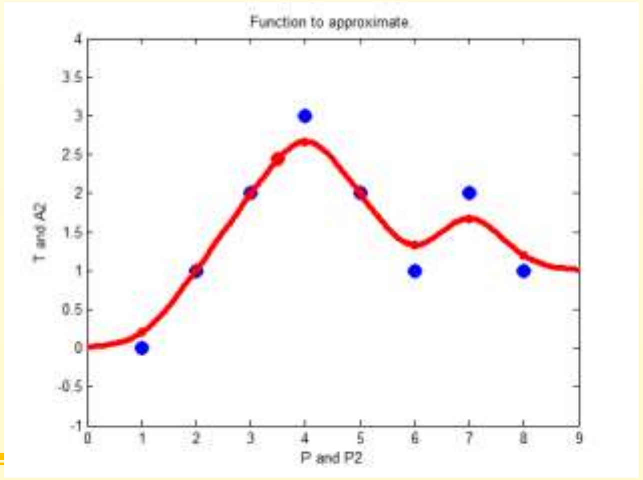
- See the paper pages 20-26

Matlab Samples

- Run Raidal_basis_demo.m



- Run General_Regression_NN.m (step by step)



newrbe, newrb, newgrnn

References

- Neural Networks and Learning Machines, Haykin, 2008
- NEURAL NETWORKS, M. Hajek, 2005
- Training RBF networks with selective backpropagation, Vakil-Baghmisheh, Neurocomputing 62 (2004) 39 – 64
- Some Slides

تنفس – جملات قصار

- **ساعتی اندیشـــیدن بهتر است از هفتاد سال عبادت.** «حضرت محمد صلی الله علیه و آله و سلم»
- **درباره هر چیزی که می‌گویی فکر کن اما درباره هر چه فکر می‌کنی، مگوی!** « بقراطیس »
- **کسی که فکر نمی کند، به ندرت دم فرو می بندد.** « نیوتن »
- **از همان لحظه که به فکر کردن خو می گیرید در راه ترقی گام بر می دارید.** « پستالوژی »

تنفس

- دوستان دو گونه اند: یکی آنان که همیشه شما را می‌خندانند، از ایشان خیری نخواهی دید. دیگر آنکه عیب شما را می‌گویند و شما را به تفکر وا می‌دارند، قدر ایشان را بدانید. « ابوالعلاء معری »

- اگر مشکلی داری، به دلیل طرز فکر توست و تنها راهی که می توانی مشکلات را برای همیشه حل کنی، این است که طرز فکرت را تغییر دهی. « وین دایر »

- آموزش را در خانواده و دانش را در جامعه می آموزند و بینش را در تفکرات تنهایی. « فردریش نیچه »