Radial Basis Function Networks

Interpolation and Approximation

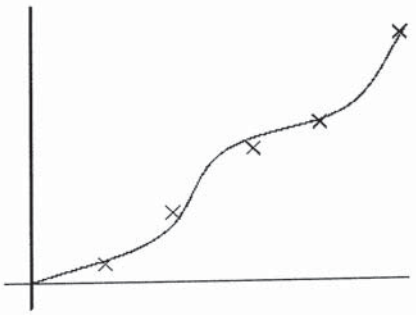
- RBF networks have their origins in techniques for performing exact *interpolation (παρεμβολή)* of a set of input-output samples
- (exact) *interpolation* – after training NN gives the exactly correct outputs for all training data
- *approximation (κατά προσέγγιση)*– NN relates well the training input vectors to the desired outputs but does not necessary produce the exactly correct outputs
- Vacations example: how much you enjoy vacations as a function of their duration

Ex.	Duration (days)	Rating (1-10)
1	4	5
2	7	9
3	9	2
4	12	6

- We would like to create an interpolation and approximation networks that predict the ratings for other durations
- Given data is called sample input-output pairs (training data)

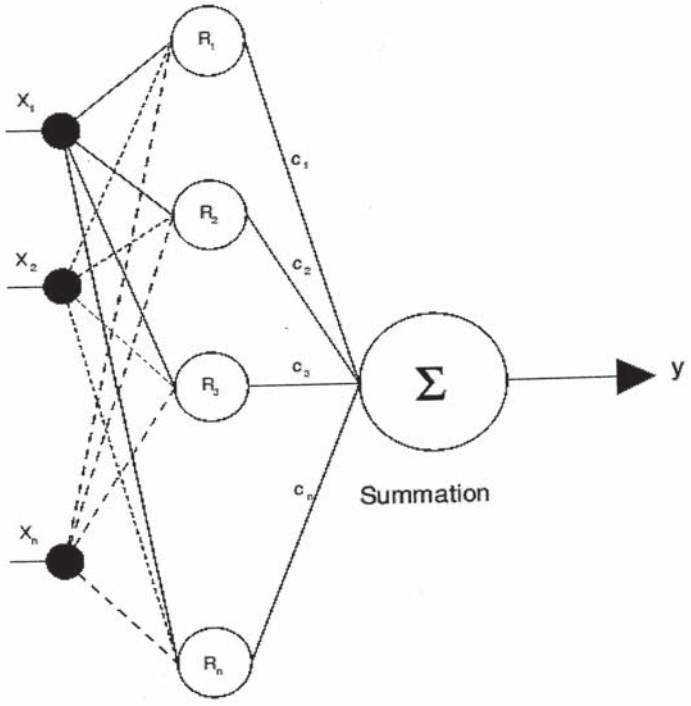
Approximation Theory

- Fitting of lines or surfaces to a set of points in a 2 or more dimensional space:
  - In 2-D space - lines are fitted to points.
  - In 3-D space - planes are fitted to points.
  - In n-D space - hyperplanes are fitted to points.
- Curve fitting is essentially the process that occurs as MLPs learn their training data.
- If there are K weights and biases in the network, the learning process thus traces out a point in (K+1) dimensional weight/error space.



The layers of an RBF network

- Input layer
  - the nodes distribute the network input vector to all the nodes in the hidden layer.
- Hidden layer
  - each node contains a centre and a function called the Basis Function. It performs a non-linear mapping from the input space into a higher dimensional space in which the patterns become linearly separable.
- Output layer
  - multiplies the output of each hidden layer node by a coefficient and sums the resulting values.
- Network output
  - the result of the summation at the output layer.



Basis Function

Fully connected layer – all neurons of the previous layer are connected with all of the next layer

Main ideas

Process performed in the hidden layer

* Idea: patterns in the input space form clusters

* if the centres of the clusters are known then the distance from the cluster centre can be measured

* the distance measure is made non-linear – so: if a pattern is in an area that is close to the cluster centre it gives a value close to 1; beyond this area the Value drops dramatically.

* Notion: this area is **radially symmetrical** around the cluster centre – so: The non-linear function becomes known as RBF.

What do RBFs consist of?

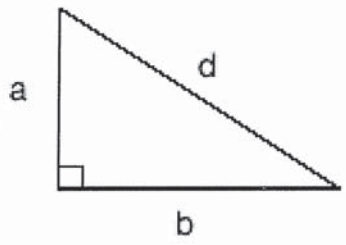
- A set of n-dimensional centres.
- A set of scalar coefficients.
- An input layer.
- A hidden layer.
- An output layer.

- ***The Parameters***
  - *A Centre (R)*
    - A vector of the same dimensionality as the network's input vector
    - This is associated with each node in the hidden layer.
  - *A Coefficient (c)*
    - A scalar value which weights the connection between the hidden layer and the output layer.

Basis functions

- Equivalent to the activation function of an MLP.
- Takes in a scalar number and returns a scalar number which is the output of the hidden layer node.
- Commonly used basis functions:
- Gaussian: $\phi(r) = \exp(-r^2/2\sigma^2)$
- *where σ represents the width of the Gaussian.*
- ***What happens in a hidden layer node?***
  - Euclidean distance between the input vector and the centre vector is found.
  - Distance produced is passed through the basis function to produce the output of the hidden layer node.

Euclidean distance



$$d = \sqrt{a^2 + b^2}$$

For a n-Dimensional vector $x = [x_1 \ldots x_n]$:

$$\| x \| = \text{Length}_{EUC} = \sqrt{\sum_{i=1}^{N} x_i^2}$$

An RBF hidden node

- $x = [x_1, x_2, ..., x_N]$ = input vector
- $R = [R_1, R_2, ..., R_N]$ = centre vector

$$\| x - R \| = d_{EUC} = \sqrt{\sum_{i=1}^{N} (X_i - R_i)^2}$$

  - *Euclidean distance between the vectors **X** and **R***

- Centre vector is subtracted from the input vector.
- The resulting distance is a scalar number which is used as the input to the basis function, to produce the hidden layer node's output (o).
  - $o = f(\| x - R \|)$

Calculating the network output

- The outputs of the hidden layer nodes are passed to the output layer. The output layer weights each hidden layer output with a coefficient and sums the resultant values.

$$y = \sum_{1}^{m} c_i o_i$$

- Where *m* is the number of nodes in the hidden layer, o_i represents the output of the i_{th} hidden layer node and c_i is the coefficient weighting the connection between the i_{th} hidden node and the output layer node.
- Output equation: $$y = \sum_{1}^{m} c_i \phi(\| x - R_i \|)$$
  - Where R_i is the centre belonging to hidden layer node i.

RBFs and Perceptrons

- RBFs have similarities to perceptrons, but are more powerful than single layer perceptrons …
  - … As a RBF network can learn problems which a single layer perceptron cannot
    - i.e. 'Hard' problems which require a Multi-Layer Perceptron.

- Perceptrons weight and sum their inputs before passing them through an activation function:

$$y = \sigma \sum_{1}^{N} x_i w_i$$

- RBFs pass each input through a basis function before weighting and summing:

$$y = \sum_{1}^{m} c_i \phi (|x - R_i|)$$

Comparison with RBFs and MLPs

- **Differences:**

Feature	MLP	RBF
Variables:	Weights & Thresholds	Centres & Coefficients
Speed of Training:	Slow	Fast (er)
	Computationally expensive	Less computation
Accuracy:	Very good	Not as good
Mathematical Description	Difficult	Easy

- **Similarities:**
  - Learn from a set of training data.
  - Ability to generalise from training data.
  - Store information in a distributed manner.
  - Can be implemented using parallel processing.

Designing an RBF

- **How to choose centres**
  - Centres chosen to represent input training data set.
  - The correct choice of centres is critical for good RBF performance.

- **The number of centres**
  - Too many or too few leads to inaccuracy.
  - Currently there are no rigorous formal methods to determine the optimum number of centres.

- **Methods for choosing centres:**
  - *Simple distribution*
    - Uniform distribution over data space
    - Gaussian distribution over data space
  - *Distribution related to the data distribution*
    - Eg. Use of clustering algorithms

Training the RBF
1. With fixed centres

$$y = \sum_{1}^{n} c_i \phi \left(\|x - R_i\| \right)$$

- Before training commences, the centres, R, are chosen. The only parameters which need to be trained are the coefficients c. All the other RBF values are known.

- The input and output training sets provide pairs of values for x and y. For each input/output, x/y, pair, an equation is produced:

$$y = c_1 \phi \left(x, R_1 \right) + c_2 \phi \left(x, R_2 \right) + \ldots + c_n \phi \left(x, R_n \right)$$

- Where:

$$\phi \left(x, R_i \right) = \phi \left(\|x - R_i\| \right)$$

Calculating the centre coefficients

- As there is usually more than one pattern in the training set a group of equations is produced:

$$
\begin{aligned}
y_1 &= c_1\phi\left(x_1, R_1\right) + c_2\phi\left(x_1, R_2\right) + \ldots + c_n\phi\left(x_1, R_n\right) \\
y_2 &= c_1\phi\left(x_2, R_1\right) + c_2\phi\left(x_2, R_2\right) + \ldots + c_n\phi\left(x_2, R_n\right) \\
y_m &= c_1\phi\left(x_m, R_1\right) + c_2\phi\left(x_m, R_2\right) + \ldots + c_n\phi\left(x_m, R_n\right)
\end{aligned}
$$

- There are usually more training pairs than there are unknown coefficients, so we have an over determined set of equations.

… In vector notation

- Each equation can be expressed in vector notation, using vector dot product to represent the summation, producing the following set of equations:

$$
\begin{aligned}
y_1 &= \phi\left(x_1, \underline{R}\right) \cdot \underline{c} \\
y_2 &= \phi\left(x_2, \underline{R}\right) \cdot \underline{c} \\
y_m &= \phi\left(x_m, \underline{R}\right) \cdot \underline{c}
\end{aligned}
$$

- These equations can be represented more simply in matrix form by:

$$
\begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_m \end{bmatrix} = \begin{bmatrix} \phi\left(x_1, \underline{R}\right) \\ \phi\left(x_2, \underline{R}\right) \\ \ldots \\ \phi\left(x_3, \underline{R}\right) \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ \ldots \\ c_n \end{bmatrix}
$$

$$
\underline{y} = \underline{A} \times \underline{c}
$$

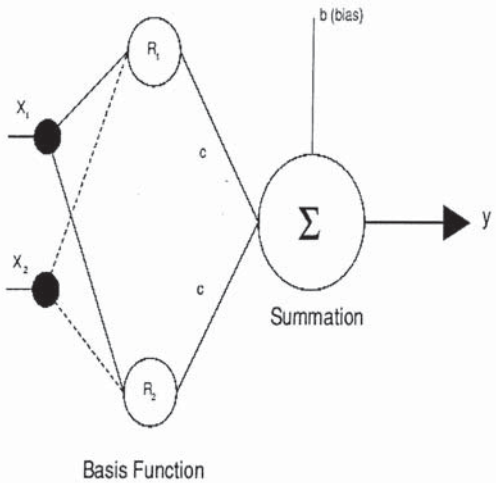
Solving for c

$$\underline{c} = \underline{y} \times \underline{A}^{-1}$$

- Solving the above equation involves finding the inverse of ***A***.
- There are a variety of techniques that can be used to produce this inverse and hence solve for the coefficients.
- The most widely used technique is singular value decomposition; this technique can deal with cases where the training patterns are closely related to each other.

Training: an example

- A simple RBF which has:
  - 2 inputs,
  - 1 output,
  - 2 centres,
  - 2 unknown coefficients (one weight and one bias).
- The XOR problem revisited:
  - 0 0 => 0
  - 0 1 => 1
  - 1 0 => 1
  - 0 0 => 0

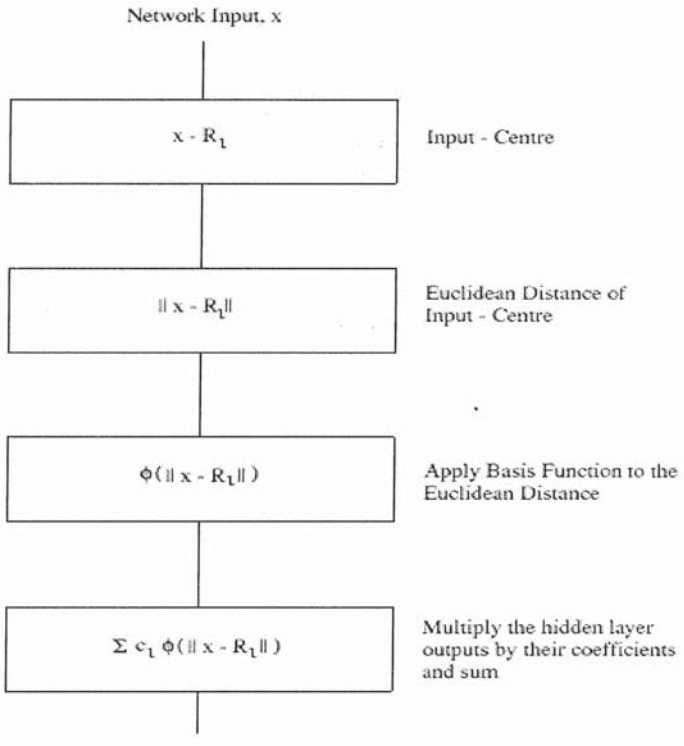


The centres

- For the characterisation of the output unit we will use the following information:
  - The output unit will employ ***weight sharing***. This is justified by the underlying symmetry of the XOR problem.
  - The output unit includes a ***bias term***. This is justified by the fact that the desired output values of the XOR function have a non zero mean.
- **The Centres**
  - The two centres are chosen as:
    - $R_1 = [1\ 1]^T$
    - $R_2 = [0\ 0]^T$

A example RBF to evaluate XOR

- The Centres:
  - The two centres are chosen as:
    - $R_1 = [1\ 1]^T$
    - $R_2 = [0\ 0]^T$
- The weights & bias:
  - $c_1 = c_2 = -1.97$
  - $b = +2.45$



Calculating the output

- The input
  - Let X = [0 1]; For XOR the desired output is [+1]
- Distance of X from R_1 [1 1]
  - SQRT $((0-1)^2 + (1-1)^2) = +1$
  - Output R_1 = Gaussian (+1) = $e^{-1} = 0.368$
- Distance of X from R_2 [0 0]
  - SQRT $((0-0)^2 + (1-0)^2) = +1$
  - Output R_2 = Gaussian (+1) = $e^{-1} = 0.368$
- Output = (0.368*(-1.97))+ (0.368*(-1.97))+ (+2.45) = 1.00008

Basis function

- Use a Gaussian function:

$$G(\| x - R \|) = e^{-\|x-R\|^2} = e^{-\sum_i (x_i - R)^2}$$

- Hence the output function is:

$$y(x) = \sum_{i=1}^{2} c\, G(\| x - R \|) + b$$

- Hence learning in this RBF is the process of finding the weight coefficient, c, and the bias term, b. Four equations can be produced, from the 4 training patterns:

$$0.0 = c\,G(\|[0\ 0] - [1\ 1]\|) + c\,G(\|[0\ 0] - [0\ 0]\|) + b$$

$$1.0 = c\,G(\|[0\ 1] - [1\ 1]\|) + c\,G(\|[0\ 1] - [0\ 0]\|) + b$$

$$1.0 = c\,G(\|[1\ 0] - [1\ 1]\|) + c\,G(\|[1\ 0] - [0\ 0]\|) + b$$

$$0.0 = c\,G(\|[1\ 1] - [1\ 1]\|) + c\,G(\|[1\ 1] - [0\ 0]\|) + b$$

Solving for *c*

Calculating the basis function value

$$G(\| [0\,0] - [1\,1] \|) = e^{-(\sqrt{((0-1)^2+(0-1)^2)})^2}$$
$$= e^{-((-1)^2+(-1)^2)}$$
$$= e^{-\sqrt{2}}$$
$$= 0.2431$$

$$0 = c(0.2431) + c(1.0000) + b$$
$$1 = c(0.3678) + c(0.3678) + b$$
$$1 = c(0.3678) + c(0.3678) + b$$
$$0 = c(1.0000) + c(0.2431) + b$$

Let $y = |0\ 1\ 1\ 0|^T$ and $c = |c\ c\ b|^T$

Let $A = \begin{vmatrix} 0.2431 & 1.0000 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.3678 & 0.3678 & 1 \\ 1.0000 & 0.2431 & 1 \end{vmatrix}$

Then $Ac = y$. But this problem is ***over-determined*** which is why A is not a square matrix.

and $c = \begin{vmatrix} -1.97 \\ -1.97 \\ +2.45 \end{vmatrix}$

These values can now be used with new input patterns to generate new outputs as required.

22

Training algorithm for RBF NN with fixed centres

1. Choose appropriate centres ***R***
2. Choose appropriate ***σ***
3. Initialise the weights/coefficients ***c*** to small random values
4. Calculate the output $\underline{y} = \underline{A} \times \underline{c} = \Phi(x_n, \underline{R}) \times \underline{c}$
5. Find the optimum weight values with $\underline{c} = \underline{y} \times \underline{A}^{-1}$

RBF Networks – (continue)

The training of RBF networks seen so far assumed fixed centres ***R***, the positions of which is usually chosen at random so as to cover the entire distribution of ***x***

- **Training RBF Networks with adjustable centres *R***
  - Use of Stochastic Gradient Descent Approach
  - In this case we can train/adjust all three sets of parameters., i.e., coefficients (or weights) ***c***, centres ***R*** and variances $\boldsymbol{\sigma}$.

Training RBF networks with adjustable centres

- **The instantaneous total error is calculated by:**

$$J[k]=\frac{1}{2}\,[\,e[k]]^2=\frac{1}{2}\,[d[k]-\sum_{k=1}^{N}c_k\,[k]\Phi(x[k],R_k[k])]^2$$

where d[k] is the desired output

- **Given that Φ is the Gaussian Function, the instantaneous total error becomes:**

$$J[k]=\frac{1}{2}\,[d[k]-\sum_{k=1}^{N}c_k\,[k]\,e^M]^2$$

where $M=-\left(\frac{||x[k],R_k[k]||^2}{\sigma_k^2\,[k]}\right)$

Training RBF networks with adjustable centres

The gradient descent equations for every one of ***c***, ***R***, and ***σ*** are given by:

$$c_k[\kappa+1] = c_k[\kappa] - \eta_c \left.\frac{\partial J[\kappa]}{\partial \boldsymbol{c}_k}\right|_{c_k=c_k[\kappa]}$$

$$R_k[\kappa+1] = R_k[\kappa] - \eta_R \left.\frac{\partial J[\kappa]}{\partial R_\kappa}\right|_{R_\kappa=R_k[\kappa]}$$

$$\sigma_k[\kappa+1] = \sigma_k[\kappa] - \eta_\sigma \left.\frac{\partial J[\kappa]}{\partial \boldsymbol{\sigma}_k}\right|_{\sigma_k=\sigma_k[\kappa]}$$

Training algorithm for RBF NN with adjustable centres

1. Choose appropriate centres ***R***
2. Choose appropriate ***σ***
3. Initialise the coefficients/weights ***c*** to small random values
4. Present an input vector, and compute the network output according to:

$$y[\kappa] = \sum_{k=1}^{N} c_k \phi(\boldsymbol{x}[\kappa], R_k, \sigma_k)$$

Training algorithm for RBF NN with adjustable centres

5. Update the network parameters according to:

$$c[\kappa+1] = c[\kappa] + \eta_c e[\kappa]\psi[\kappa]$$

$$R_k[\kappa+1] = R_k[\kappa] + \eta_R \frac{e[\kappa]c_k[\kappa]}{\sigma_k^2[\kappa]} \phi\{x[\kappa], R_k[\kappa], \sigma_k[\kappa]\}[x[\kappa] - R_k[\kappa]]$$

$$\sigma_k[\kappa+1] = \sigma_k[\kappa] + \eta_\sigma \frac{e[\kappa]c_k[\kappa]}{\sigma_k^3[\kappa]} \phi\{x[\kappa], R_k[\kappa], \sigma_k[\kappa]\} \| x[\kappa] - R_k[\kappa] \|^2$$

where:

$$\psi[\kappa] = [\phi\{x[\kappa], R_1[\kappa], \sigma_1[\kappa]\}, \phi\{x[\kappa], R_2[\kappa], \sigma_2[\kappa]\}, \ldots, \phi\{x[\kappa], R_N[\kappa], \sigma_N[\kappa]\}]^T$$

$$e[\kappa] = d[\kappa] - y[\kappa]$$

6. Stop if the network has converged; else go back to step 4.

Training algorithm for RBF NN with adjustable centres

- Note that this training algorithm is less complex than BP because there is only one layer of adjustable weights; backpropagation of errors is not required

- The learning rates η_c, η_R and η_σ can be set to different values

RADIAL BASIS FUNCTIONS (continue)

Vacations example (revisited): how much you enjoy vacations as a function of their duration

Ex.	Duration (days)	Rating (1-10)
1	4	5
2	7	9
3	9	2
4	12	6

We would like to create an interpolation and approximation networks that predict the ratings for other durations

Interpolation RBF Network – use of RBFs with fixed centres:

- **Can be solved by a RBF network with *k* neurons in the hidden layer and 1 output neuron**
  - **In this case we could make each neuron in the hidden layer to respond to one sample input vector**
  - **Centres would therefore be the input vectors in this case; σ is the width of the Gaussians; if small – each input vector will have only local influence, if wide – global influence**
  - **The output layer adds the weighted outputs of the hidden layer**

31

Equations for the Vacation Example

Given σ, we can compute the weights

*** A system of linear equations**
- *** Which are the unknowns? How many are they?**
- *** How many equations?**

$$y(\mathbf{x}_1) = w_1 e^{-\frac{\|\mathbf{x}_1-\mathbf{x}_1\|^2}{2\sigma}} + w_2 e^{-\frac{\|\mathbf{x}_1-\mathbf{x}_2\|^2}{2\sigma}} + w_3 e^{-\frac{\|\mathbf{x}_1-\mathbf{x}_3\|^2}{2\sigma}} + w_4 e^{-\frac{\|\mathbf{x}_1-\mathbf{x}_4\|^2}{2\sigma}}$$

$$y(\mathbf{x}_2) = w_1 e^{-\frac{\|\mathbf{x}_2-\mathbf{x}_1\|^2}{2\sigma}} + w_2 e^{-\frac{\|\mathbf{x}_2-\mathbf{x}_2\|^2}{2\sigma}} + w_3 e^{-\frac{\|\mathbf{x}_2-\mathbf{x}_3\|^2}{2\sigma}} + w_4 e^{-\frac{\|\mathbf{x}_2-\mathbf{x}_4\|^2}{2\sigma}}$$

$$y(\mathbf{x}_3) = w_1 e^{-\frac{\|\mathbf{x}_3-\mathbf{x}_1\|^2}{2\sigma}} + w_2 e^{-\frac{\|\mathbf{x}_3-\mathbf{x}_2\|^2}{2\sigma}} + w_3 e^{-\frac{\|\mathbf{x}_3-\mathbf{x}_3\|^2}{2\sigma}} + w_4 e^{-\frac{\|\mathbf{x}_3-\mathbf{x}_4\|^2}{2\sigma}}$$

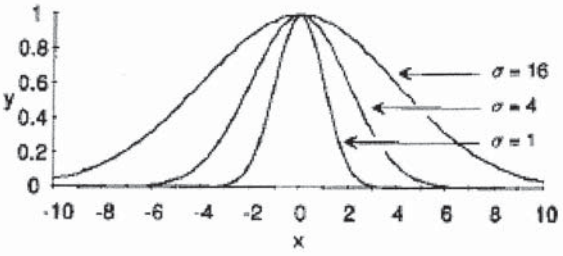
$$y(\mathbf{x}_4) = w_1 e^{-\frac{\|\mathbf{x}_4-\mathbf{x}_1\|^2}{2\sigma}} + w_2 e^{-\frac{\|\mathbf{x}_4-\mathbf{x}_2\|^2}{2\sigma}} + w_3 e^{-\frac{\|\mathbf{x}_4-\mathbf{x}_3\|^2}{2\sigma}} + w_4 e^{-\frac{\|\mathbf{x}_4-\mathbf{x}_4\|^2}{2\sigma}}$$

32

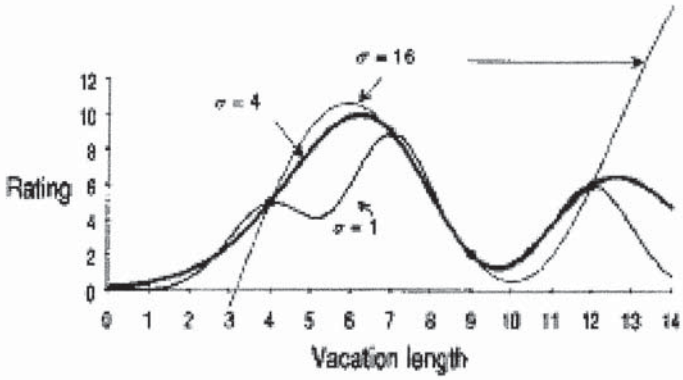
Vacation Example - Solving the Equations

* Solutions for 3 different σ:

σ	w1	w2	w3	w4
1	4.90	8.84	0.73	5.99
4	0.87	13.93	-9.20	8.37
16	-76.50	236.49	-237.77	87.55



* The vacation rating approximation functions



- Small σ => bumpy interpolations
- Large σ => extend the influence of each sample
- Too large σ => too global influence of each sample

Summary: How to Create an Interpolation RBF Net

- For each given sample (training set example), create a neuron in the first layer (i.e. the hidden layer) centred on the sample input vector
- Create a system of linear equations
  - Choose σ
  - Compute the Euclidean distance between the sample input and each of the centres
  - Compute the Gaussian functions of each distance
  - Multiply the Gaussian function by the corresponding weight/coefficient of the neuron
  - Equate the sample output with the sum of the weighted Gaussian functions of distance
- Solve the system equations for the weights/coefficients
- There is no training of the weights but solving of equations
- The network can be used to predict the output of new examples

Creation of an Approximation RBF net

- If there are many samples, the number of neurons in the hidden layer of the interpolation nets will be high
- Solution: Approximate the unknown function with a smaller number of neurons that are somehow representative
  - The simplest way – pick up a few random samples to serve as centres
- Such networks are called approximation nets
  - As the number of neurons in the hidden layer is lower than the number of samples, a set of weights/coefficients such that the network produces the exact correct output for all samples does not exist
  - However, a set of weights/coefficients that result in a reasonable approximation for all samples can be found
  - One way is to use the gradient descent to minimize the error – *RBF networks with adjustable centres*

How to Create RBF Approximation Nets

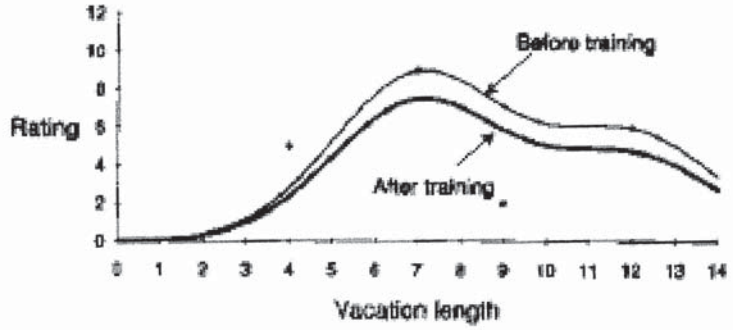
- For a few samples, create an approximation RBF net
- Select learning rates ($\eta_{w/c}$, η_R, η_σ)
- Until performance is satisfactory (error below a minimum threshold or max number of epochs reached):
  - For all sample inputs
  - Compute the resulting outputs
  - Compute the adjustment of weights/coefficients, the centres and the standard deviations
  - Add up the parameter (*c/w*, *R*, σ) changes for all the sample inputs and update the parameters (batch mode) or update the parameters after each example (online update)

Approximation RBF Net for the Vacation Example

Adjustment of the weights/coefficients only

- 2 neurons only in the hidden layer
- initialized to example 2 (7 days) and 4 (12 days) (i.e., these are taken as the centres for the two hidden neurons)
- Learning rate $\eta_{w/c} = 0.1$, $\sigma = 4$
- The weights after 100 epochs:

	w1	w2	R1	R2
Initial values	8.75	5.61	7	12
Final values	7.33	4.47	7	12



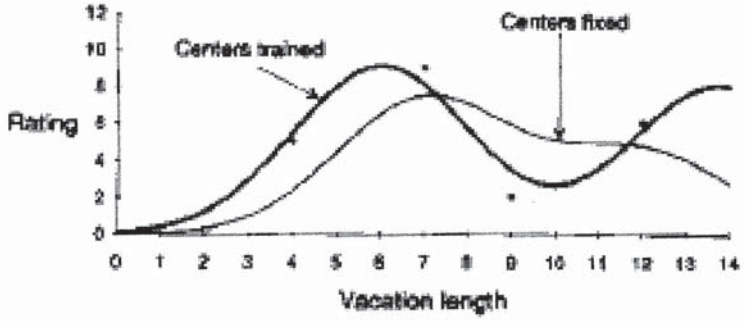
37

Approximation RBF Net for the Vacation Example

Adjustment of the weights/coefficients and centres

* Not only weights/coefficients but also the centres and standard deviations can be adjusted
* How much (for the centres)? Find the derivatives of the error function with respect to the centre coordinates (for centre adjustment, see relevant equation from previous lecture)
* The weights after 100 epochs:

	w1	w2	R1	R2
Initial values	8.75	5.61	7	12
Final values	9.13	8.06	6	13.72

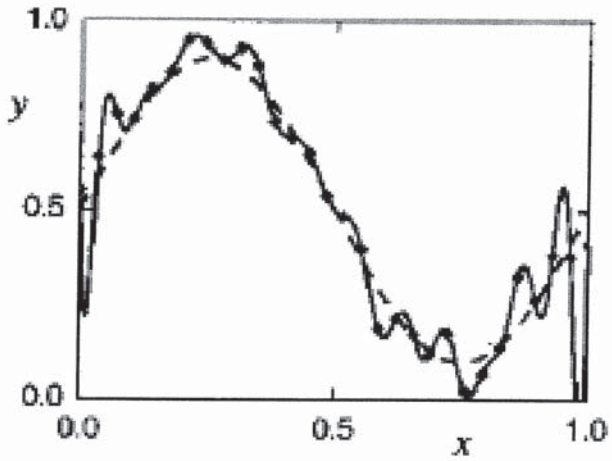


* Adjustment of both weights and centres provides a better approximation than adjustment of either of them alone

38

RBFs for Interpolation and Approximation – Summary Example

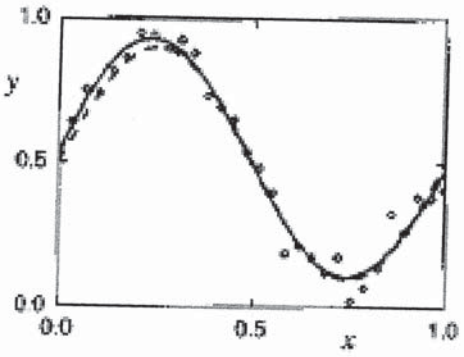
- Example taken from Bishop (1995), "NNs for Pattern Recognition"
- A set of 30 data points was generated by sampling the function $y = 0.5 + 0.4\sin(2\pi x)$, shown by the dashed curve, and adding Gaussian noise with standard deviation 0.05



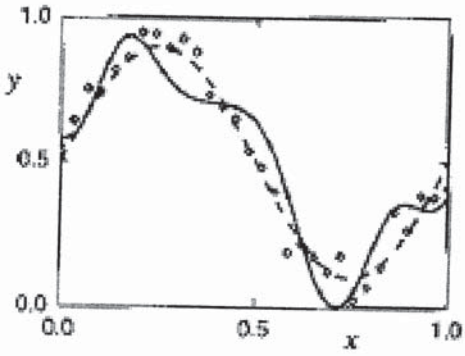
- RBF network for exact interpolation has been created
- Solid curve - the interpolation function which resulted using $\sigma = 0.067$ (=~ twice the average spacing between the data points)
- The weights between the RBF and the output later were found by solving the equations

Summary Example (continue)

- Approximation RBF networks with 3 different widths have been created

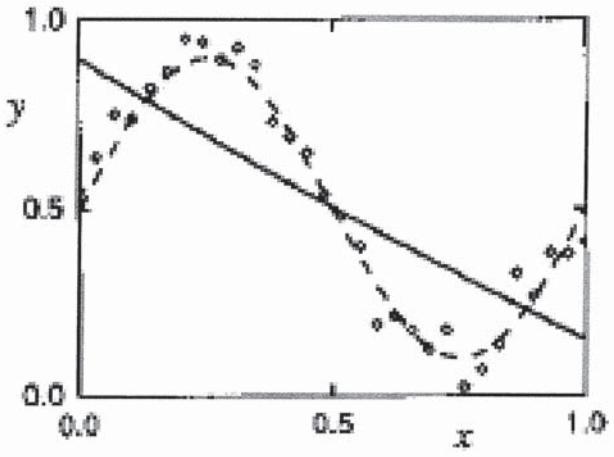


- RBF neurons = 5 (significantly smaller than the number of data points)
- RBF centres were set to a random subset of the given data points
- $\sigma = 0.4$ (=~ twice the average spacing between centres)
- The weights were found by training
- Good fit



- As above, but $\sigma = 0.08$
- The resulting network function is insufficiently smooth and gives a poor representation of the underlying function which generated the data

Summary Example (continue)



- As on the previous slide, but with $\sigma = 10$
- The resulting network function is over-smoothed (the receptive fields of the RBF overlap too much) and again gives a poor representation of the underlying function which generated the data

- *Conclusion*: To get good generalisation the width should be larger than the average distance between adjacent input vectors but smaller than the distance across all input space

RBFs – Useful Notes

Width of the Gaussian (or Standard Deviation σ)

***For an interpolation task*:**
Small σ → bumpy interpolations

Large σ → extend the influence of each sample

Very large σ → too global influence of each sample

***For an approximation task*:**
To get a good generalisation, the width should be larger than the average distance between adjacent vectors, but smaller than the distance across all input space.

RBF NNs for Classification

Cover's Theorem on the Separability of Patterns

"A complex classification problem cast in *high dimensional* space nonlinearly is more likely to be *linearly separable* than in a *low* dimensional space" (Cover, 1965). This includes two ingredients:

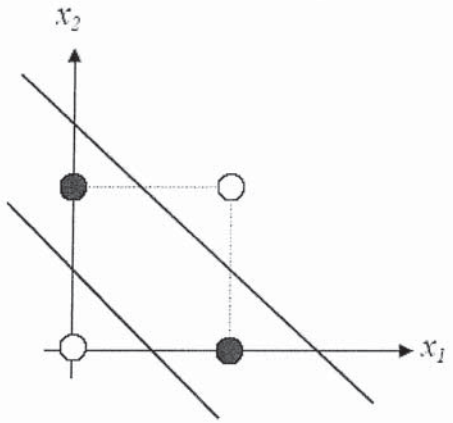
1. **Nonlinear formulation of hidden functions**
2. **High dimensionality of the hidden space compared to the input space (determined by the number of hidden neurons)**

In some cases, however, the use of nonlinear mapping (i.e., point 1) may be sufficient to produce linear separability without having to increase the dimensionality of the hidden-neuron space (see example below)

The XOR problem revisited

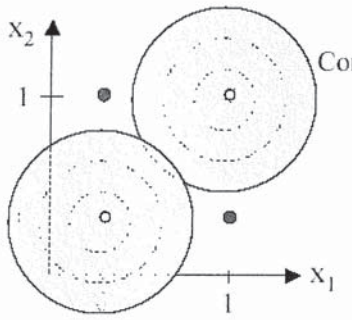
We are already familiar with the non-linearly separable XOR problem:

p	x_1	x_2	t
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0



We know that Single Layer Perceptrons with step activation functions cannot generate the right outputs, because they are only able to form a single decision boundary. To deal with this problem we needed to change the activation function and introduce a non-linear hidden layer to form the Multi Layer Perceptron (MLP).

RBF NNs on XOR problem



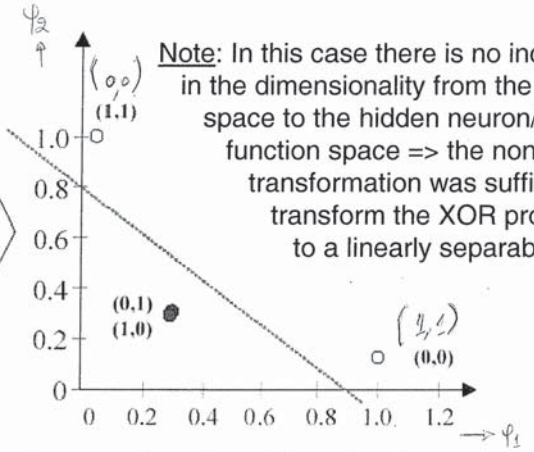
Consider the nonlinear functions to map the input vector $\mathbf{x}$ to the φ_1- φ_2 space

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

$$\varphi_1(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{t}_1\|^2} \qquad \mathbf{t}_1 = [1\ 1]^T$$

$$\varphi_2(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{t}_2\|^2} \qquad \mathbf{t}_2 = [0\ 0]^T$$

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



Note: In this case there is no increase in the dimensionality from the input space to the hidden neuron/basis function space => the nonlinear transformation was sufficient to transform the XOR problem to a linearly separable one

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

45

The XOR Problem Output Weights

In this case we just have one output *y(x)*, with one weight w_j to each hidden unit *j*, and one bias - θ. This gives us the network's input-output relation for each input pattern *x*

$$y(x) = w_1\varphi_1(x) + w_2\varphi_2(x) - \theta$$

Then, if we want the outputs $y(x^p)$ to equal the targets t^p, we get the four equations

$$1.0000w_1 + 0.1353w_2 - 1.0000\theta = 0$$
$$0.3678w_1 + 0.3678w_2 - 1.0000\theta = 1$$
$$0.3678w_1 + 0.3678w_2 - 1.0000\theta = 1$$
$$0.1353w_1 + 1.0000w_2 - 1.0000\theta = 0$$

Three are different, and we have three variables, so we can easily solve them to give

$$w_1 = w_2 = -2.5018\ ,\quad \theta = -2.8404$$

This completes our "training" of the RBF network for the XOR problem.

46

Creating RBF Networks for Classification

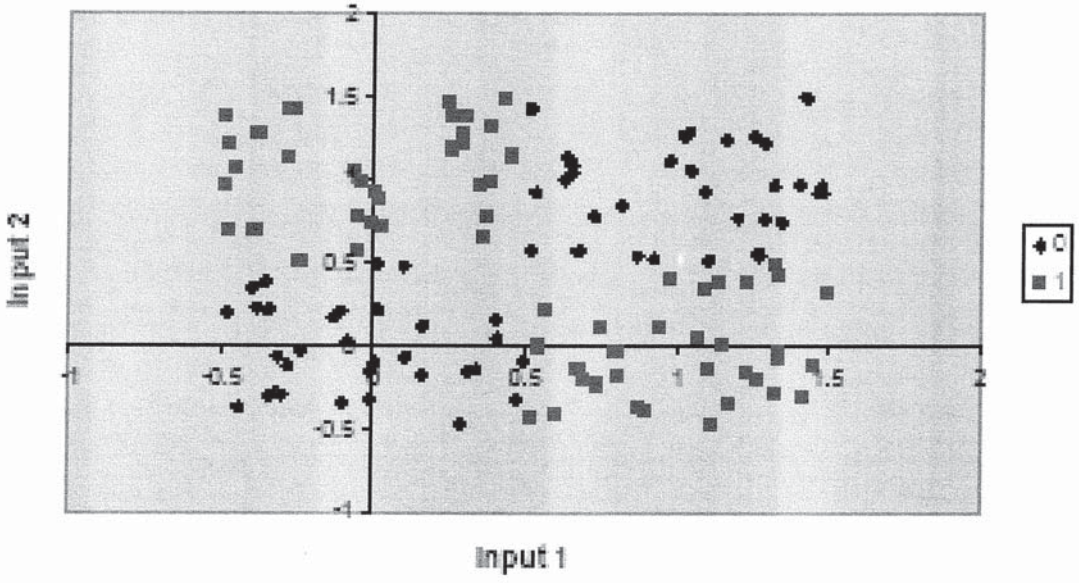
- 1. Select the locations of the centers R_i
  - Choose them randomly from the training data or
  - Use a clustering algorithm to find them, e.g. SOM
- 2. Fit the RBFs, i.e. set the width of the Gaussians
  - Set their width using the p-nearest neighbour method
    - σ_i of a centre i is the root mean squared distance from that centre to the p nearest centres
    - p is set by trial and error, typically p=2
    - Small p => narrower Gaussians (may be expected to emphasize the nature of the training data, great risk of overtraining)
    - Higher p => will broaden the Gaussians, may be expected to suppress the distinction between clusters
  - Other heuristics: $\sigma_i = \sqrt{d_i}$ here d_i is the distance to the nearest centre
- 3. Gradient descent to train the linear layer
  - Typically one output neuron for each class => i.e. binary encoding of the targets

Classification Example – Noisy XOR Problem

Noisy XOR truth table:

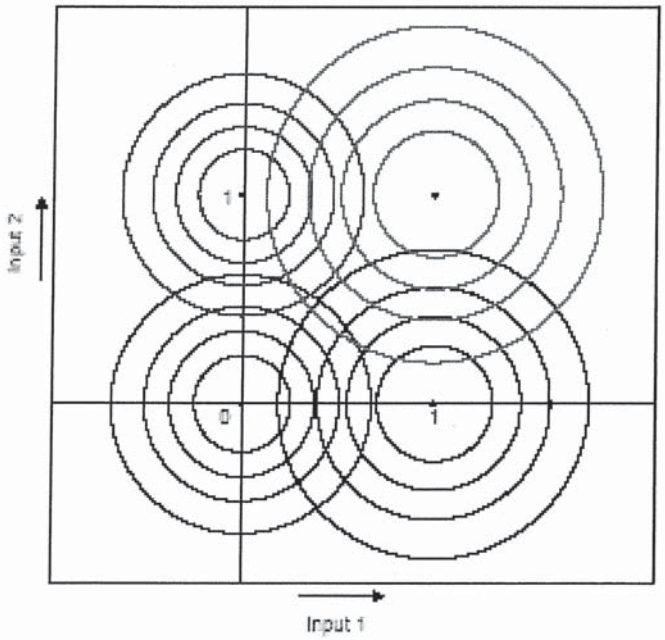
Input 1	Input 2	Output
<0.5	<0.5	0
<0.5	>0.5	1
>0.5	<0.5	1
>0.5	>0.5	0

Correct Classifications



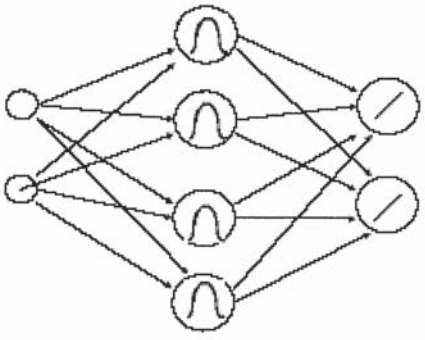
Noisy XOR Problem (continue)

1. Clustering the data using Kohonen SOM
2. Locating the centres of the RBF by setting them to the centres of the clusters
3. Determining their widths
4. The corresponding RBF net:
   * 2 input and 4 RBF hidden neurons



Noisy XOR Problem (continue)

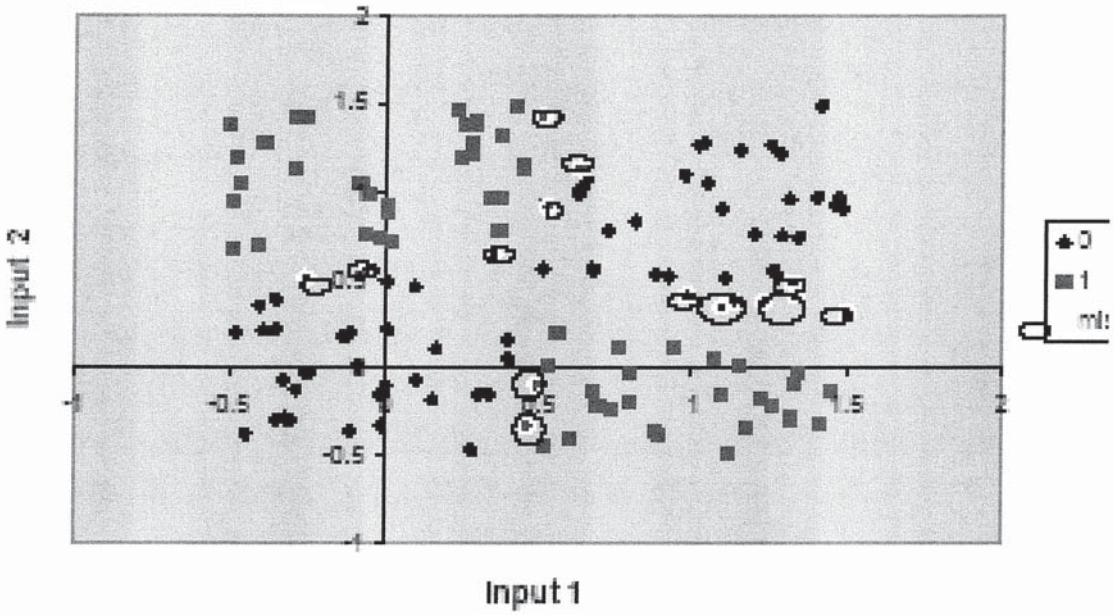
5. Create an output neuron for each class and use binary encoding for the targets (1 0 for class 1 and 0 1 for class 2)



6. Initialize the weights between hidden and output layer to small random values
7. Train the weights between the hidden and output layer using the gradient descent algorithm

Noisy XOR Problem (continue)

Classification by the RBF network (misclassifications are circled)



- **Where are the misclassifications?**
- **Why RBF nets are not perfect fit for this data?**

51

RBF networks and MLPs - Similarities

Similarities - both MLP and RBF NNs are:

- **examples of non-linear, layered, feed-forward networks**
- **universal approximators**
  - **Universality of RBF NNs: Any function can be approximated to arbitrary small error by an RBF NN given there are enough RBF neurons**
  - **Hence, there always exist an RBF network capable of accurately mimicking a specified MLP, and vice versa**

24 52

RBF networks and MLPs – Differences

- An RBF network has a simple architecture (2 layers: nonlinear and linear) with single hidden layer; MLP may have 1 or 2 hidden layers and an active output layer
- MLPs may have a complex pattern of connectivity (not necessarily fully connected), RBF is a fully connected network.
- The hidden layer of an RBF net is non-linear, and the output layer is linear. The hidden and output layers of an MLP used as a classifier are nonlinear.
- Hidden and output neurons in a MLP share a common neuron model (nonlinear transformation of linear combination). Hidden neurons in RBF nets are quite different and serve a different purpose than the output neurons.

~~25~~ ~~54~~
53

RBF networks and MLPs – Differences (cont)

- **Computation at hidden neurons**

  * The argument of the activation function of a hidden neuron in RBF nets computes the Euclidean distance between the input vector and the centre of the neuron.

  * The activation function of a hidden neuron in MLP computes the inner product of the input vector and the weight vector of that neuron.

- **Representation formed (in the space of hidden neurons activations with respect to the input space)**

  * An MLP is said to form a distributed representation since for a given input vector many hidden neurons will typically contribute to the output value.

  * RBF nets form a local representation as for a given input vector only a few hidden neurons will have significant activation and contribute to the output.

~~25~~ 54

RBF networks and MLPs – Differences (cont)

Differences in how the weights are determined

• In MLP - usually at the same time as a part of a single global training strategy involving supervised learning.

• A RBF net is typically trained in 2 stages with the RBFs being determined first by unsupervised technique using the given data alone, and the weights between the hidden and output layer being found by fast linear supervised methods.

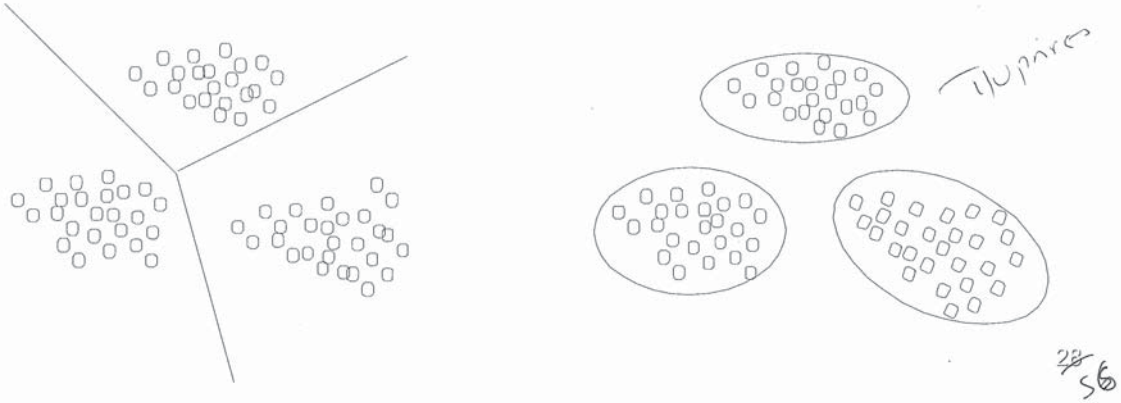
RBF networks and MLPs – Differences (cont)

Differences in the separation of the data

• Hidden neurons in MLP define hyper-planes in the input space (a)

• RBF nets fit each class using kernel functions (b)

(a) (b)



RBF networks and MLPs – Differences (cont)

Number of neurons

* MLPs typically have less number of of neurons than RBF NNs. This is because they respond to large regions of the input space.

* RBF nets may require many RBF neurons to cover the input space (larger number of input vectors and larger ranges of input vectors => more RBF neurons are needed.) However, they are capable of fast learning and have reduced sensitivity to the order of presentation of training data.

Radial Basis Function Networks - Learning Algorithm for variable centres

1 Notation

Weights (coefficients)	w	Centre	R
Target Output	t	Hidden Node/Centre Iterator	h
Actual Output	y	Number of Centres	m
Input Pattern	x	Output Iterator	o
Input Dimension Iterator	i	Number of Outputs	k
Input Dimension	d	Standard Deviation	σ
Number of Inputs	n	Learning Rates	η_R, η_σ, η_w
Input Pattern Iterator	p	Bias unit	b

2 Algorithm Pseudocode

1. Choose the number (m) and initial coordinates of the centres (R) of the RBF functions.
2. Choose the initial value of the spread parameter (σ) for each centre (R).
3. Initialise the weights/coefficients (w) to small random values [-1,1].
4. For each epoch (e)
5. For each input vector/pattern ($x^{(p)}$)
6. Calculate the output (y) of each output node (o) using eq. 1.
7. Update the network parameters (w, R, σ) using eqs. 6, 7, 8, 9.
8. end for (p = n)
9. end for (e = total epochs)

Note: Steps 1 and 2 can be performed using a clustering algorithm such as Kohonen SOMs.

3 Equations

1. The output of node o for pattern p is:

$$y_o^{(p)} = w_{bo} + \sum_{h=1}^{m} w_{ho}\phi(x^{(p)}, R_h, \sigma_h)$$

2. Gaussian function:

$$\phi(x^{(p)}, R_h, \sigma_h) = exp\left(-\frac{||x^{(p)} - R_h||^2}{2\sigma_h^2}\right)$$

3. Euclidean distance:

$$||x^{(p)} - R_h||^2 = \sum_{i=1}^{d}\left(x_i^{(p)} - R_{hi}\right)^2$$

4. Total Error:

$$E = \frac{1}{2}\sum_{p=1}^{n}\sum_{o=1}^{k}\left[\varepsilon_o^{(p)}\right]^2$$

5. Error of output neuron o for pattern p:

$$\varepsilon_o^{(p)} = t_o^{(p)} - y_o^{(p)}$$

6. Weight/coefficient update equation (from hidden unit/centre h to output node o):

$$w_{ho}^{'} = w_{ho} + \eta_w \varepsilon_o^{(p)}\phi(x^{(p)}, R_h, \sigma_h)$$

7. Weight/coefficient update equation (from bias unit b to output node o):

$$w_{bo}^{'} = w_{bo} + \eta_w \varepsilon_o^{(p)}$$

8. Centre update equation (ithcoordinate of centre h):

$$R_{hi}^{'} = R_{hi} + \eta_R \sum_{o=1}^{k} \varepsilon_o^{(p)} w_{ho}\phi(x^{(p)}, R_h, \sigma_h)\frac{x_i^{(p)} - R_{hi}}{\sigma_h^2}$$

9. Standard Deviation/width σ of Gaussian of centre h:

$$\sigma_h^{'} = \sigma_h + \eta_\sigma \sum_{o=1}^{k} \varepsilon_o^{(p)} w_{ho}\phi(x^{(p)}, R_h, \sigma_h)\frac{||x^{(p)} - R_h||^2}{\sigma_h^3}$$

Radial Basis Function Neural Network Demo – Fixed Centres

https://campusvirtual.ull.es/ocw/file.php/55/applets/rbfPrg/html/index.html

Radial Basis Function Neural Network Demo – Variable Centres

http://myweb.ncku.edu.tw/~stli/www/teach/ke/RBF/demo1/

Radial Basis Function Network

Introduction

This program demonstrates some function approximation capabilities of a Radial Basis Function Network.

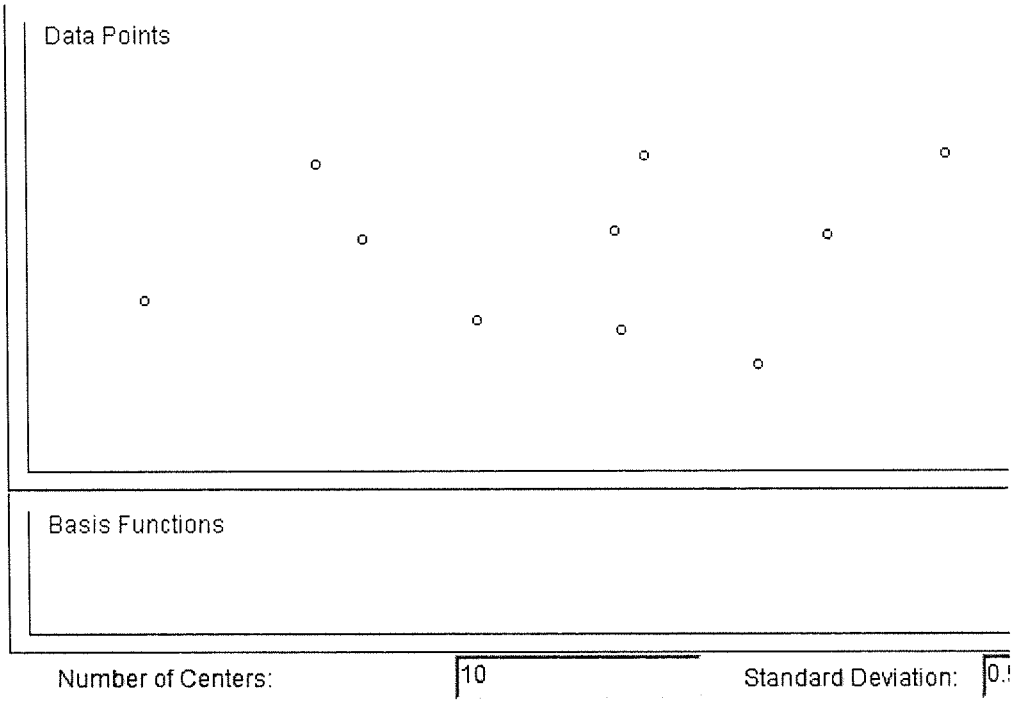
The user supplies a set of training points which represent some "sample" points for some arbitrary curve. Next, the user specifies the number of equally spaced gaussian centers and the variance for the network. Using the training samples, the weights multiplying each of the gaussian basis functions arecalculated using the pseudo-inverse (yielding the minimum least-squares solution). The resulting network is then used to approximate the function between the given "sample" points.

Credits

This Java applet (□ 1996 - Jesse W. Hong, Massachusetts Institute of Technology) was integrated as is into this web page with the agreement of the author. Please send comments about this applet directly to *jesse@mit.edu*.

Instructions

- Use the first mouse button to place training points in the upper section. Try to space them approximately equally, and place more points than number of centers being used.
- Adjust the number of centers and the standard deviation (width of Gaussian) for the Gaussians used in the approximation.
- Click on "Redraw" to redraw the basis functions and the training points. After any adjustment to the number of centers or variance, click on this button to show the new basis functions.
- Click on "Go!" to train the network.
- The "Reset" button erases all the training points and lets you start again.
- Status messages are shown at the bottom of the applet.



Comments

After learning, the scaled plots of each of the gaussians is shown in green in the upper graph. The resulting approximation of the curve which is the sum of all the scaled gaussians is shown in red.

Try to make sure that your data points are equally spaced. If they are not and you have a lot of centers, then in between some of the data points, the fitted curve may go off screen. Just add a new point in these regions and click on "Go!" again.

Play around with the number of centers and the standard deviation and see how the smoothness and accuracy of the approximation is affected. E.g. make the standard deviation 0.25, place 10 equally spaced data points, click on "Go!", then make the standard deviation 1.0, and click on "Go!" again. You can also play with the number of data points, etc.

Questions

1. Put down 32 data points so that the data looks like a noisy version of some function. Take the width of the Gaussians as 1.5.
   a) As a first step, try to do an interpolation, that is, use 32 Gaussians. Type 'go' and look at the result. Then reduce the number of centers to 16, then to 8, and finally to 4. What is the result?
   b) Repeat the same sequence but with a width of 1.0 and 0.5. What is the result?
2. Put down about 30 data points in two clusters, one for low x-values and another one for high x-values. Take 10 centers.
   a) Do a fit with standard deviation 1.5, then with 1.0. Is the result reasonable?
   b) Now add one extra data point in the middle between the two clusters. Is the result getting better? Try to optimize parameters (number of centers and standard deviation).