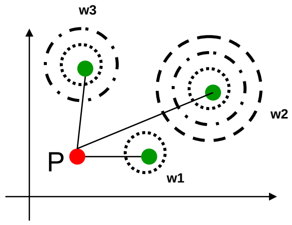
RBF

Radial-Basis Function Networks

- A function is radial (RBF) if its output depends on (is a non-increasing function of) the distance of the input from a given stored vector.
- RBFs represent local receptors, as illustrated below, where each 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.



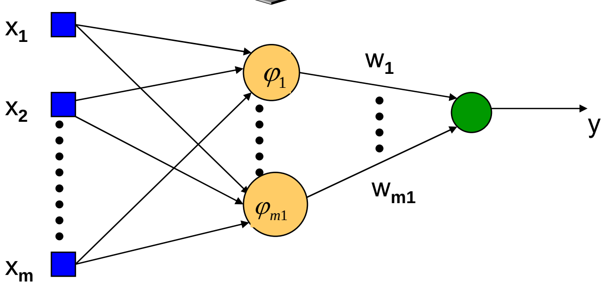
The vector P is "interpolated" using the three vectors; each vector gives a contribution that depends on its weight and on its distance from the point P. In the picture we have

$$w1 < w3 < w2$$

NN 5 1

RBF

RBF ARCHITECTURE



- One hidden layer with RBF activation functions

$$\varphi_1 \ldots \varphi_{m1}$$

- **Output layer with linear activation function.**

$$y = w_1\varphi_1(\| x - t_1 \|) + \ldots + w_{m1}\varphi_{m1}(\| x - t_{m1} \|)$$

$$\| x - t \| \text{distance of } x = (x_1, \ldots, x_m) \text{ from vector } t$$

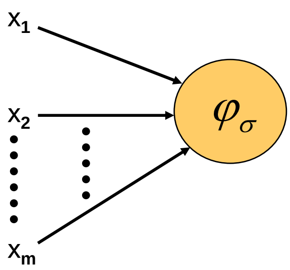
NN 5 2

RBF

HIDDEN NEURON MODEL

- Hidden units: use radial basis functions

  $\varphi_\sigma(|| x - t||)$ the output depends on the distance of the input x from the center t



$\varphi_\sigma(|| x - t||)$

t is called center
σ is called spread
center and spread are parameters

RBF

Hidden Neurons

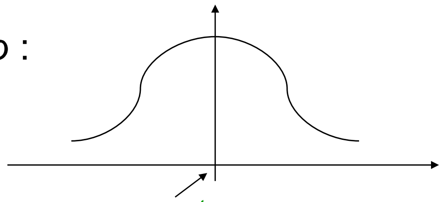
- A hidden neuron is more sensitive to data points near its center.

- For Gaussian RBF this sensitivity may be tuned by adjusting the spread σ, where a larger spread implies less sensitivity.

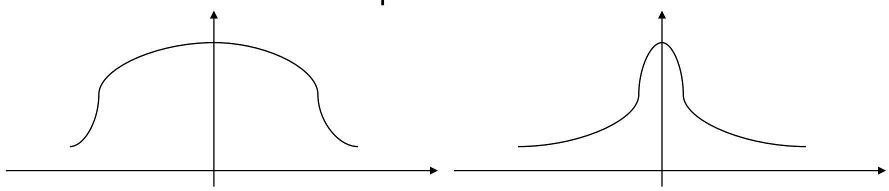
RBF

Gaussian RBF φ

φ :



σ is a measure of how spread the curve is:



NN 5 5

RBF

Interpolation with RBF

The interpolation problem:

Given a set of N different points $\{x_i \in \Re^m, i = 1 \cdots N\}$ and a set of N real numbers $\{d_i \in \Re \ , i = 1 \cdots N\}$, find a function $F : \Re^m \Rightarrow \Re$ that satisfies the interpolation condition: $F(x_i) = d_i$

If $F(x) = \sum_{i=1}^{N} w_i \varphi(\|x - x_i\|)$ we have:

$$\begin{bmatrix} \varphi(\|x_1 - x_1\|) & \dots & \varphi(\|x_1 - x_N\|) \\ & \dots & \\ \varphi(\|x_N - x_1\|) & \dots & \varphi(\|x_N - x_N\|) \end{bmatrix} \begin{bmatrix} w_1 \\ \dots \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ \dots \\ d_N \end{bmatrix} \Rightarrow \Phi w = d$$

NN 5 6

RBF

Types of φ

Micchelli's theorem:

Let $\{x_i\}_{i=1}^{N}$ a set of distinct points in $\Re^m$ Then the N-by-N interpolation matrix Φ , whose ji-th element is $\varphi_{ji} = \varphi\left(\|x_j - x_i\|\right)$ is nonsingular.

- Multiquadrics: Inverse multiquadrics:

$$\varphi(r) = (r^2 + c^2)^{1/2} \qquad \varphi(r) = \frac{1}{(r^2 + c^2)^{1/2}} \qquad c > 0 \qquad r = \| x - t \|$$

- Gaussian functions (most used):

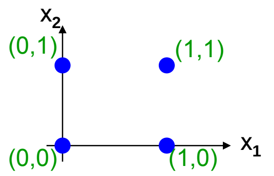
$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \qquad \sigma > 0$$

NN 5 7

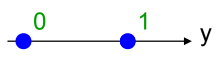
RBF

Example: the XOR problem

- Input space:



- Output space:



- Construct an RBF pattern classifier such that:
  (0,0) and (1,1) are mapped to 0, class C1
  (1,0) and (0,1) are mapped to 1, class C2

NN 5 8

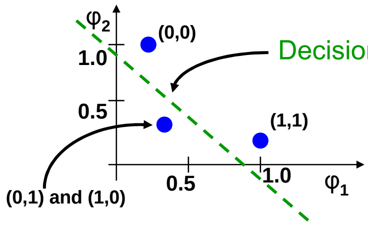
RBF

Example: the XOR problem

- In the feature (hidden layer) space:

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

$$\varphi_2(\| x - t_2 \|) = e^{-\|x-t_2\|^2} \quad \text{with } t_1 = (1,1) \text{ and } t_2 = (0,0)$$



- When mapped into the feature space < φ_1 , φ_2 > (hidden layer), C1 and C2 become *linearly separable*. So a linear classifier with $\varphi_1(x)$ and $\varphi_2(x)$ as inputs can be used to solve the XOR problem.

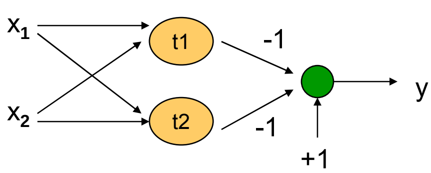
NN 5 9

RBF

RBF NN for the XOR problem

$$\varphi_1(\| x - t_1 \|) = e^{-\|x-t_1\|^2} \quad \text{with } t_1 = (1,1) \text{ and } t_2 = (0,0)$$

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



$$y = -e^{-\|x-t_1\|^2} - e^{-\|x-t_2\|^2} + 1$$

$$\text{If } y > 0 \text{ then class 1 otherwise class 0}$$

NN 5 10

RBF

RBF network parameters

- **What do we have to learn for a RBF NN with a given architecture?**
  - The centers of the RBF activation functions
  - the spreads of the Gaussian RBF activation functions
  - the weights from the hidden to the output layer
- Different learning algorithms may be used for learning the RBF network parameters. We describe three possible methods for learning centers, spreads and weights.

NN 5 11

RBF

Learning Algorithm 1

- **Centers: are selected at random**
  - **centers** are chosen randomly from the training set
- **Spreads:** are chosen by **normalization**:

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

- Then the activation function of hidden neuron i becomes:

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

NN 5 12

RBF

Learning Algorithm 1

- **Weights:** are computed by means of the **pseudo-inverse method.**
  - **For an example** (x_i, d_i) **consider the output of the network**

$$y(x_i) \approx w_1\varphi_1(\| x_i - t_1 \|) + \ldots + w_{m1}\varphi_{m1}(\| x_i - t_{m1} \|)$$

  - **We would like** $y(x_i) = d_i$ **for each example, that is**

$$w_1\varphi_1(\| x_i - t_1 \|) + \ldots + w_{m1}\varphi_{m1}(\| x_i - t_{m1} \|) \approx d_i$$

NN 5 13

RBF

Learning Algorithm 1

- This can be re-written in matrix form for one example

$$\left[\varphi_1(\| x_i - t_1 \|) \ldots \varphi_{m1}(\| x_i - t_{m1} \|)\right] \begin{bmatrix} w_1 \\ \ldots \\ w_{m1} \end{bmatrix} = d_i$$

and

$$\begin{bmatrix} \varphi_1(\| x_1 - t_1 \|) \ldots \varphi_{m1}(\| x_1 - t_{m1} \|) \\ \ldots \\ \varphi_1(\| x_N - t_1 \|) \ldots \varphi_{m1}(\| x_N - t_{m1} \|) \end{bmatrix} \begin{bmatrix} w_1 \\ \ldots \\ w_{m1} \end{bmatrix} = [d_1 \ldots d_N]^T$$

for all the examples at the same time

NN 5 14

RBF

Learning Algorithm 1

let
$$\Phi = \begin{bmatrix} \varphi_1(\| x_1 - t_1 \|) & \dots & \varphi_{m1}(\| x_N - t_{m1} \|) \\ & \dots & \\ \varphi_1(\| x_N - t_1 \|) & \dots & \varphi_{m1}(\| x_N - t_{m1} \|) \end{bmatrix}$$

then we can write
$$\Phi \begin{bmatrix} w_1 \\ \dots \\ w_{m1} \end{bmatrix} = \begin{bmatrix} d_1 \\ \dots \\ d_N \end{bmatrix}$$

So the unknown vector w is the solution of the linear systems

$$\Phi w = d \quad \text{or} \quad \Phi^T \Phi w = \Phi^T d$$

NN 5 15

Pseudoinverse matrix

If we define $\Phi^+ \equiv \left(\Phi^T \Phi\right)^{-1} \Phi^T$ as the pseudo-inverse of the matrix Φ we can obtain the weights using the following formula

$$[w_1 \dots w_{m1}]^T = \Phi^+ [d_1 \dots d_N]^T$$

The evaluation of Φ^+ usually requires to store in memory the entire matrix Φ.

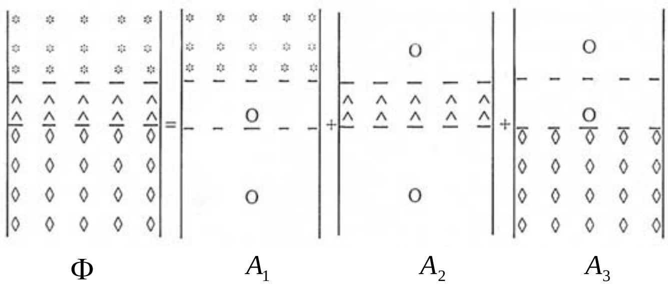
NN 5 16

Pseudoinverse matrix

A good idea is to divide Φ in Q subsets each made by an arbitrary number a_i of rows (i=1…..Q)

Then we build a set of new matrices named A_i; in A_i the only set of rows different from zero is the i-th subset of matrix Φ so we have

$$\Phi = \sum_{i=1}^{Q} A_i$$



NN 5 17

Pseudoinverse matrix

If we replace $\Phi = \sum_{i=1}^{Q} A_i$ in the definition of Φ^+ we have:

$$w = \left(\Phi^T\Phi\right)^{-1}\Phi^T d = \left(\sum_{j,i=1}^{Q} A_j^T A_i\right)^{-1}\left(\sum_{j=1}^{Q} A_j^T\right)d$$

We define

$$H = \sum_{j,i=1}^{Q} A_j^T A_i$$

NN 5 18

Pseudoinverse matrix

Observations:

- every product $A_j^T A_i$ is a square M-by-M matrix
- because of the particular form of the matrices A_i, every product $A_j^T A_i$ with $i \neq j$ is the null M-by-M matrix
- if we define a set of reduced matrices $\hat{A}_i$ made only out of the rows different from zero of A_i, we have $A_i^T A_i \equiv \hat{A}_i^T \hat{A}_i$.

NN 5 19

Pseudoinverse matrix

In order to obtain $H = \sum_{i=1}^{Q} \hat{A}_i^T \hat{A}_i$

1) store $\hat{A}_i$ (dim $\hat{A}_i = a_i \times$ M)

2) compute $\hat{A}_i^T$ and therefore $\hat{A}_i^T \hat{A}_i$

3) add this product to a list that after Q steps will contain the whole matrix H.

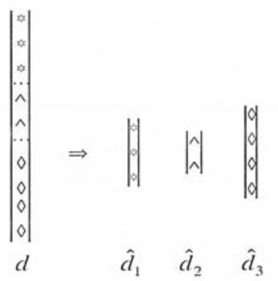
We store in memory the matrices $\hat{A}_i$ one at a time; the products $\hat{A}_i^T \hat{A}_i$ can also be computed by different processing units

NN 5 20

Pseudoinverse matrix

In order to obtain $w = H^{-1}\left(\sum_{j=1}^{Q} A_j^T\right)d$ we use a similar technique: since

$H^{-1}\left(\sum_{j=1}^{Q} A_j^T\right)d = H^{-1}\sum_{j=1}^{Q} A_j^T d$, we introduce Q reduced vectors $\hat{d}_i$.



Each vector $\hat{d}_i$ is made only by the rows

$$\left(\sum_{k=1}^{i-1} a_k\right)+1,\ \left(\sum_{k=1}^{i-1} a_k\right)+2,\ \ldots\ldots\left(\sum_{k=1}^{i-1} a_k\right)+a_i$$

of the vector d so

$$A_j^T d \equiv \hat{A}_j^T \hat{d}_j \quad \text{and} \quad w = H^{-1}\sum_{j=1}^{Q} \hat{A}_j^T \hat{d}_j$$

NN 5 21

Pseudoinverse matrix

Finally we compute w:

1) store $\hat{A}_j$
2) compute $\hat{A}_j^T$ and therefore $\hat{A}_j^T \hat{d}_j$
3) add the term $\hat{A}_j^T \hat{d}_j$ to a list
4) multiply the obtained sum by H^{-1}

It is possible to use Q processing units, each performing one step to completely parallelize the procedure.

The execution of the entire algorithm thus requires to store twice in memory the whole set of matrices $\hat{A}_j$

NN 5 22

RBF

Learning Algorithm 1: summary

1. Choose the centers randomly from the training set.

2. Compute the spread for the RBF function using the normalization method.

3. Find the weights using the pseudo-inverse method.

RBF

Learning Algorithm 2: Centers

- **clustering algorithm for finding the centers**
  1 **Initialization**: $t_k(0)$ random $k = 1, \dots, m_1$
  2 **Sampling**: draw x from input space
  3 **Similarity matching**: find index of center closer to x

$$k(x) = \arg\min_k \|x(n) - t_k(n)\|$$

  4 **Updating**: adjust centers

$$t_k(n+1) = \begin{cases} t_k(n) + \eta[x(n) - t_k(n)] & \text{if } k = k(x) \\ t_k(n) & \text{otherwise} \end{cases}$$

  5 **Continuation**: increment *n* by *1*, goto *2* and continue until no noticeable changes of centers occur

Learning Algorithm 2: summary

RBF

- ***Hybrid Learning Process:***
  - **Clustering** for finding the centers.
  - Spreads chosen by normalization.
  - **LMS algorithm (see Adaline)** for finding the weights.

NN 5 25

Learning Algorithm 3

RBF

- **Apply the gradient descent method for finding centers, spread and weights, by minimizing the (instantaneous) squared error** $E = \frac{1}{2}(y(x) - d)^2$
- **Update for:**

centers $$\Delta t_j = -\eta_{t_j} \frac{\partial E}{\partial t_j}$$

spread $$\Delta \sigma_j = -\eta_{\sigma_j} \frac{\partial E}{\partial \sigma_j}$$

weights $$\Delta w_{ij} = -\eta_{ij} \frac{\partial E}{\partial w_{ij}}$$

NN 5 26

RBF

Comparison with FF NN

RBF-Networks are used for regression and for performing complex (non-linear) pattern classification tasks.

Comparison between RBF networks and FFNN:

- Both are examples of *non-linear layered feed-forward* networks.
- Both are *universal approximators*.

NN 5 27

RBF

Comparison with multilayer NN

- Architecture:
  - RBF networks have one *single* hidden layer.
  - FFNN networks may have *more* hidden layers.
- Neuron Model:
  - In RBF the neuron model of the hidden neurons is *different* from the one of the output nodes.
  - Typically in FFNN hidden and output neurons share a *common neuron model*.
  - The hidden layer of RBF is *non-linear*, the output layer of RBF is *linear*.
  - Hidden and output layers of FFNN are usually *non-linear*.

NN 5 28

RBF

Comparison with multilayer NN

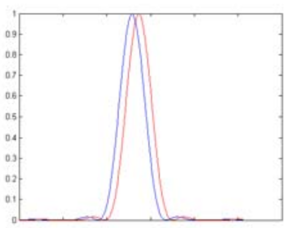
- Activation functions:
  - The argument of activation function of each hidden neuron in a RBF NN computes the *Euclidean distance* between input vector and the center of that unit.
  - The argument of the activation function of each hidden neuron in a FFNN computes the *inner product* of input vector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

NN 5 29

Example: Astronomical image processing

We used neural networks to verify the similarity of real astronomical images to predefined reference profiles, so we analyse realistic images and deduce from them a set of parameters able to describe their discrepancy with respect to the ideal, non-aberrated image.

The focal plane image, in an ideal case, is the Airy function (blue line):



$$I(r) = \frac{P\,S_p}{\lambda^2 R^2}\left[\frac{2\,J_1(\pi r)}{\pi r}\right]^2$$

Usually the image is perturbed by aberrations associated to the characteristics of real-world instruments (red line represents the perturbed image)

NN 5 30

Example: Astronomical image processing

A realistic image is associated to the Fourier transform of the aperture function:

$$I(r,\phi)=\frac{P\,S_p}{\pi^2\lambda^2R^2}\left|\int_0^1 d\rho\int_0^{2\pi} d\theta\,\rho\, e^{i\Phi(\rho,\theta)}e^{-i\pi r\rho\cos(\theta-\phi)}\right|^2$$

which includes the contributions of the classical (Seidel) aberrations:

$$\Phi(\rho,\theta)=\frac{2\pi}{\lambda}\left[A_s\rho^4+A_c\rho^3\cos\theta+A_a\rho^2\cos^2\theta+A_d\rho^2+A_t\rho\cos\theta\right]$$

A_s : Spherical aberration A_c : Coma A_a: Astigmatism

A_d : Field curvature (d = defocus) A_t : Distortion (t = tilt)

NN 5 31

Example: Astronomical image processing

- The maximum range of variation considered is $\pm 0.3\lambda$ for the training set and $\pm 0.25\lambda$ for the test set
- To ease the computational task the image is encoded using the first moments:

$$\mu_{2,0}\quad \mu_{0,2}\quad \mu_{0,3}\quad \mu_{0,4}\quad \mu_{1,1}\quad \mu_{2,1}\quad \mu_{1,2}\quad \mu_{3,1}\quad \mu_{2,2}\quad \mu_{1,3}$$

where:

$$\mu_{nm}=\frac{\iint dx\,dy\left(\frac{x-\mu_x}{\sigma_x}\right)^n\left(\frac{y-\mu_y}{\sigma_y}\right)^m\cdot I(x,y)}{\iint dx\,dy\,I(x,y)}$$

NN 5 32

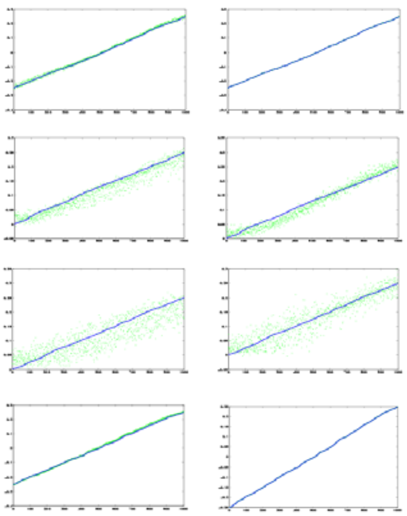
Example: Astronomical image processing

We performed this task using
- a FNN neural network (with 60 hidden nodes) and
- a RBF neural network (with 175 hidden nodes)

Usually the former requires less internal nodes, but more training iterations than the latter

The training required 1000 iterations (FNN) and 175 iterations (RBF); both optimised networks provide reasonably good results in terms of approximation of the desired unknown function relating aberrations and moments.

Example: Astronomical image processing



Performances of the sigmoidal (left) and radial (right) neural networks; from top to bottom, coma, astigmatism, defocus and distortion are shown. The blue (solid) line follows the test set targets, whereas the green dots are the values computed by the networks.