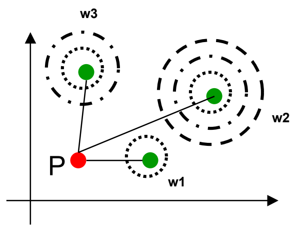
Radial-Basis Function Networks

RBF

- A function is radial basis (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.

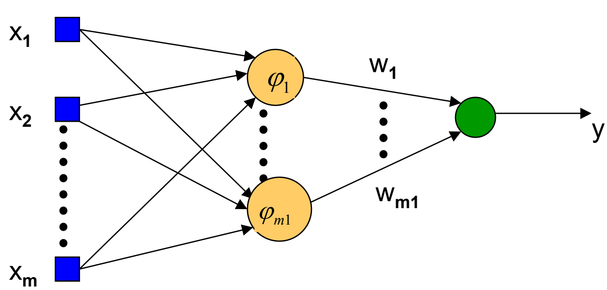


The vector P is "interpolated" using the three vectors; each vector gives a contribution that depends on its weight and on its distance from the point P. In the picture we have

$$w1 < w3 < w2$$

RBF ARCHITECTURE

RBF



- **One hidden layer with RBF activation functions**

$$\varphi_1 \dots \varphi_{m1}$$

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

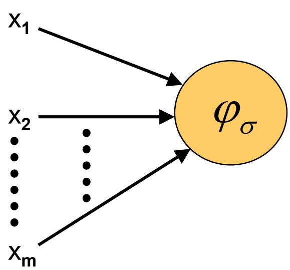
$$y = w_1\varphi_1(\| x - t_1 \|) + \dots + w_{m1}\varphi_{m1}(\| x - t_{m1} \|)$$

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

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

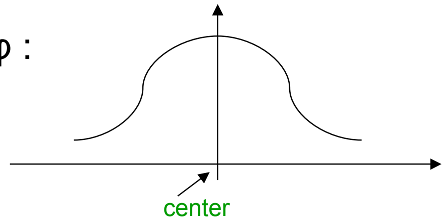
HIDDEN NEURON MODEL

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

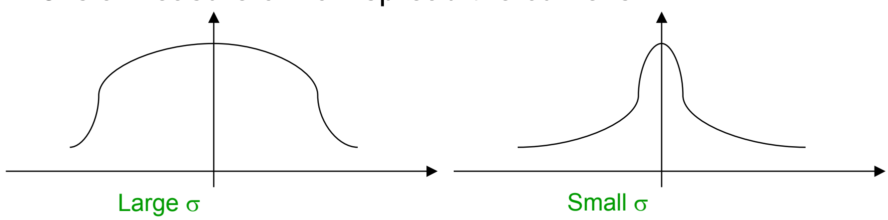
- For Gaussian RBF this sensitivity may be tuned by adjusting the spread σ, where a larger spread implies less sensitivity.

Gaussian RBF φ

φ :



σ is a measure of how spread the curve is:



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:

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

Types of φ

Micchelli's theorem:

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

- Multiquadrics:

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

- Inverse multiquadrics:

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

$$c > 0$$

$$r = \| x - t \|$$

- Gaussian functions (most used):

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

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.

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:

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

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**

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

Learning Algorithm 1

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

$$[\varphi_1(\| x_i - t_1 \|) \ldots \varphi_{m1}(\| x_i - t_{m1} \|)][w_1 \ldots w_{m1}]^T = d_i$$

and

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

for all the examples at the same time

Learning Algorithm 1

let

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

then we can write

$$\Phi \begin{bmatrix} w_1 \\ \ldots \\ w_{m1} \end{bmatrix} = \begin{bmatrix} d_1 \\ \ldots \\ d_N \end{bmatrix}$$

If Φ^+ is the pseudo-inverse of the matrix Φ we obtain the weights using the following formula

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

Learning Algorithm 1: summary

RBF

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.

Learning Algorithm 2: Centers

RBF

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

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

  4 **Updating**: adjust centers

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

  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.

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

Comparison with multilayer 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*.

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*.

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.