Radial Basis Function NN

Typical Applications of NN

- Pattern Classification

$$l = f(\mathbf{x}) \qquad \begin{array}{l} \mathbf{x} \in X \subset R^m \\ l \in C \subset N \end{array}$$

- Function Approximation

$$\mathbf{y} = f(\mathbf{x}) \qquad \begin{array}{l} \mathbf{x} \in X \subset R^n \\ \mathbf{y} \in Y \subset R^m \end{array}$$

- Time-Series Forecasting

$$\mathbf{x}(t) = f(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \mathbf{x}_{t-3}, \ldots)$$

Radial Basis Function NN

Radial Basis Functions are feed-forward networks consisting of
- A hidden layer of radial kernels and
- An output layer of linear neurons

The two layers in an RBF carry entirely different roles [Haykin, 1999]
- The hidden layer performs a **non-linear transformation** of input space
- The resulting hidden space is typically of **higher dimensionality** than the input space
- The output layer performs **linear regression** to predict the desired targets

Why use a non-linear transformation followed by a linear one?
- Cover's theorem on the separability of patterns:

"A complex pattern-classification problem cast in a high-dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space"

Introduction to Radial Basis Function Networks

The Radial Basis Function Networks

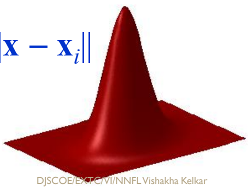
Radial Basis Functions

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

Three parameters for a radial function:

$$\phi_i(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{x}_i\|)$$

- Center $\mathbf{x}_i$
- Distance Measure $r = \|\mathbf{x} - \mathbf{x}_i\|$
- Shape ϕ



Typical Radial Functions

- Gaussian

$$\phi(r) = e^{-\frac{r^2}{2\sigma^2}} \quad \sigma > 0 \quad \text{and} \quad r \in \Re$$

- Hardy Multiquadratic

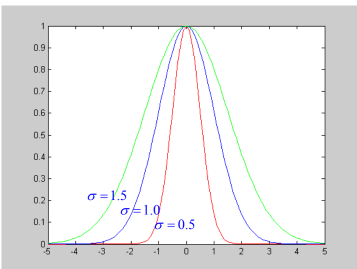
$$\phi(r) = \sqrt{r^2 + c^2}/c \quad c > 0 \quad \text{and} \quad r \in \Re$$

- Inverse Multiquadratic

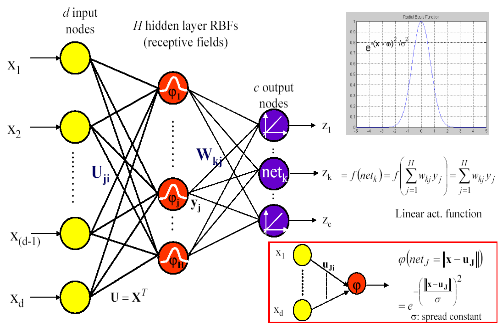
$$\phi(r) = c/\sqrt{r^2 + c^2} \quad c > 0 \quad \text{and} \quad r \in \Re$$

Gaussian Basis Function (σ=0.5,1.0,1.5)

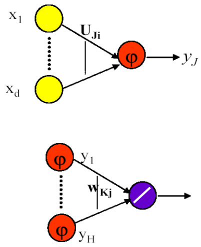
$$\phi(r) = e^{-\frac{r^2}{2\sigma^2}} \quad \sigma > 0 \quad \text{and} \quad r \in \Re$$



RBF Networks



RBF Networks



Euclidean Norm

$$\varphi\left(net_J = \|\mathbf{x}-\mathbf{u_J}\|\right) = e^{-\left(\frac{\|\mathbf{x}-\mathbf{u_J}\|}{\sigma}\right)^2}$$

σ: spread constant

$$z_K = f(net_K) = f\left(\sum_{j=1}^{H} w_{Kj} y_j\right) = \sum_{j=1}^{H} w_{Kj} y_j$$

Unknowns: $\mathbf{u_{ji}}$, $\mathbf{w_{kj}}$, σ

Back to the XOR Problem

- Recall that in the XOR problem, there are four patterns (points), namely, (0,0),(0,1),(1,0),(1,1), in a two dimensional input space.
- We would like to construct a pattern classifier that produces the output 0 for the input patterns (0,0),(1,1) and the output 1 for the input patterns (0,1),(1,0).

Example

- An often quoted example which shows how the RBF network can handle a non-linearly separable function is the exclusive-or problem.
- One solution has 2 inputs, 2 hidden units and 1 output.
- The centres for the two hidden units are set at c1 = 0,0 and c2 = 1,1, and the value of radius σ is chosen such that $2\sigma^2 = 1$.

Back to the XOR Problem – Cont’d

- We will define a pair of Gaussian hidden functions as follows:

$$\phi_1(x) = e^{-\|x-t_1\|^2}, \quad t_1=[1,1]^T$$

$$\phi_2(x) = e^{-\|x-t_2\|^2}, \quad t_2=[0,0]^T$$

Example

- The inputs are x, the distances from the centres squared are r, and the outputs from the hidden units are φ.
- When all four examples of input patterns are shown to the network, the outputs of the two hidden units are shown in the following table.

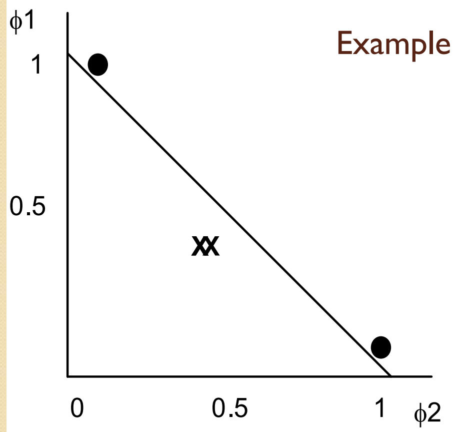
Example

x_1	x_2	r_1	r_2	φ_1	φ_2
0	0	0	2	1	0.1353
0	1	1	1	0.3678	0.3678
1	0	1	1	0.3678	0.3678
1	1	2	0	0.1353	1

Example

- Next Fig. shows the position of the four input patterns using the output of the two hidden units as the axes on the graph - it can be seen that the patterns are now linearly separable.
- This is an ideal solution - the centres were chosen carefully to show this result.
- Methods generally adopted for learning in an RBF network would find it impossible to arrive at those centre values - later learning methods that are usually adopted will be described.

Example



Function Approximation via Basis Functions and RBF Networks

Using nonlinear functions, we can convert a nonlinearly separable problem into a linearly separable one.

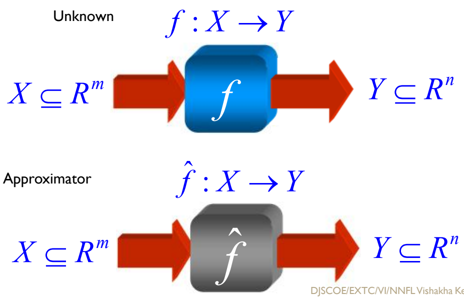
- From a function approximation perspective, this is equivalent to implementing a complex function (corresponding to the nonlinearly separable decision boundary) using simple functions (corresponding to the linearly separable decision boundary)

Implementing this procedure using a network architecture, yields the RBF networks, if the nonlinear mapping functions are *radial basis functions*.

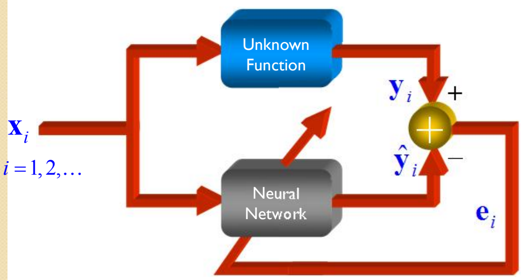
Radial Basis Functions:

- **Radial:** Symmetric around its center
- **Basis Functions:** Also called **kernels**, a set of functions whose linear combination can generate an arbitrary function in a given function space.

Function Approximation



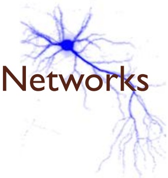
Supervised Learning



Neural Networks as Universal Approximators

- Feedforward neural networks with a single hidden layer of sigmoidal units are capable of approximating uniformly any continuous multivariate function, to any desired degree of accuracy.
- Like feedforward neural networks with a single hidden layer of sigmoidal units, it can be shown that RBF networks are universal approximators.

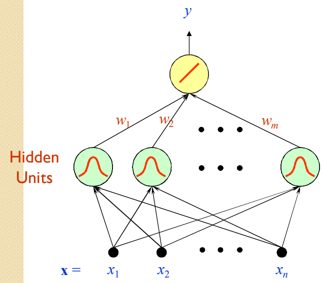
Introduction to Radial Basis Function Networks



The Model of
Function Approximator

Radial Basis Function Networks as Universal Aproximators

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$



With sufficient number of radial-basis-function units, it can also be a universal approximator.

Linear Models

$$f(\mathbf{x}) = \sum_{i=1}^{m} \overbrace{w_i}^{\text{Weights}} \underbrace{\phi_i(\mathbf{x})}_{\text{Fixed Basis Functions}}$$

Example Linear Models

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

- Polynomial

$$f(x) = \sum_i w_i x^i \qquad \phi_i(x) = x^i, \quad i = 0, 1, 2, \ldots$$

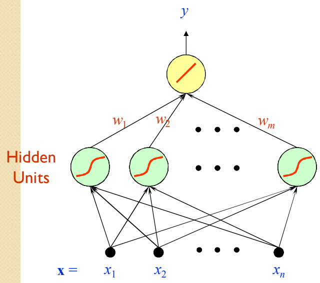
- Fourier Series

$$f(x) = \sum_k w_k \exp\left(j2k\omega_0 x\right)$$

$$\phi_k(x) = \exp\left(j2k\omega_0 x\right), \quad k = 0, 1, 2, \ldots$$

Single-Layer Perceptrons as Universal Aproximators

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$



With sufficient number of sigmoidal units, it can be a universal approximator.

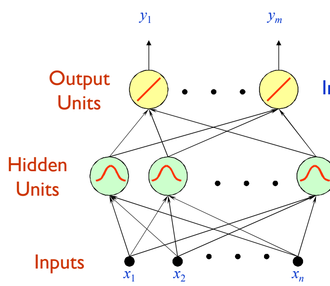
Non-Linear Models

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

$$f(\mathbf{x}) = \sum_{i=1}^{m} \overbrace{w_i}^{\text{Weights}} \underbrace{\phi_i(\mathbf{x})}_{\text{Adjusted by the Learning process}}$$

The Topology of RBF

As a function approximator $\mathbf{y} = \mathbf{f}(\mathbf{x})$



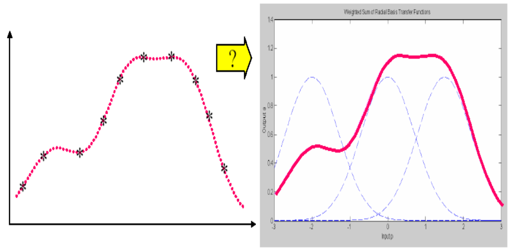
Introduction to Radial Basis Function Networks

RBFN's for

Function Approximation

Function Approximation with Radial Basis Functions

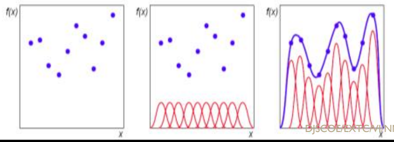
RBF Networks approximate functions using **(radial) basis functions** as the building blocks.



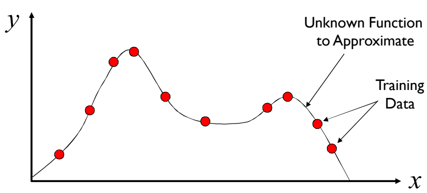
Exact Interpolation

Radial Basis Function approach (Powel 1987):

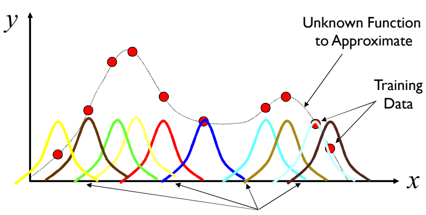
- Use a set of N basis functions of the form $\phi(||x-x^n||)$, one for each point, where $\phi(.)$ is some non-linear function.
- Output: $h(x) = \Sigma_n w_n \phi(||x-x^n||)$



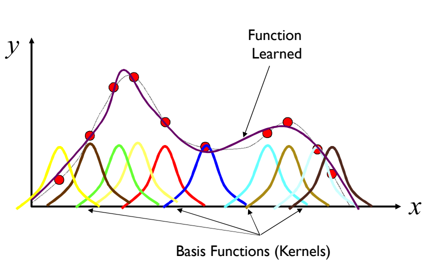
The idea



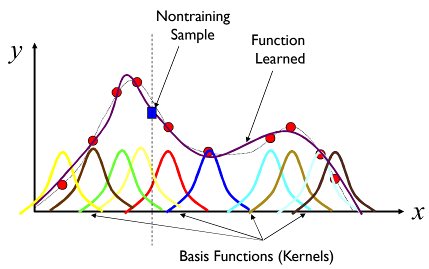
The idea



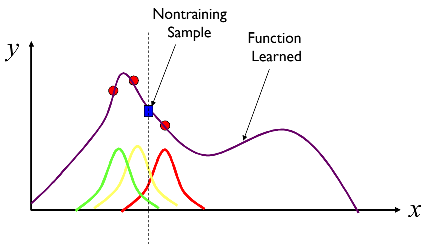
The idea $y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$



The idea $y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$

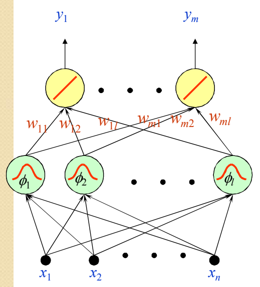


The idea $y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$



Learning RBF networks

What to Learn?



- Weights w_{ij}'s
- Centers μ_j's of ϕ_j's
- Widths σ_j's of ϕ_j's
- Number of ϕ_j's – Model Selection

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

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

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

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

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

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

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

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

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.

Learning Algorithm 2
Training hidden layer

- The hidden layer in a RBF network has units which have weights that correspond to the vector representation of the centre of a cluster.
- These weights are found either using a traditional clustering algorithm such as the k-means algorithm, or adaptively using essentially the Kohonen algorithm.

Learning Algorithm 2
Training hidden layer

- In either case, the training is unsupervised but the number of clusters that you expect, k, is set in advance. The algorithms then find the best fit to these clusters.
- The k -means algorithm will be briefly outlined.
- Initially k points in the pattern space are randomly set.

Learning Algorithm 2
Training hidden layer

- Then for each item of data in the training set, the distances are found from all of the *k* centres.
- The closest centre is chosen for each item of data - this is the initial classification, so all items of data will be assigned a class from 1 to *k*.
- Then, for all data which has been found to be class 1, the average or mean values are found for each of co-ordinates.

Learning Algorithm 2
Training hidden layer

- These become the new values for the centre corresponding to class 1.
- Repeated for all data found to be in class 2, then class 3 and so on until class *k* is dealt with - we now have *k* new centres.
- Process of measuring the distance between the centres and each item of data and re-classifying the data is repeated until there is no further change – i.e. the sum of the distances monitored and training halts when the total distance no longer falls.

Learning Algorithm 2: summary

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

Radial Basis Function Networks as Universal Aproximators

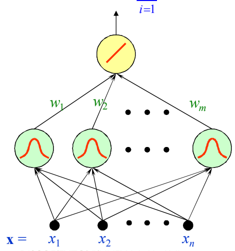
Training set $\mathcal{T} = \left\{\left(\mathbf{x}^{(k)}, y^{(k)}\right)\right\}_{k=1}^{p}$

Goal $y^{(k)} \approx f\left(\mathbf{x}^{(k)}\right)$ for all k

$$\min E = \frac{1}{2}\sum_{k=1}^{p}\left[y^{(k)} - f\left(\mathbf{x}^{(k)}\right)\right]^2$$

$$= \frac{1}{2}\sum_{k=1}^{p}\left[y^{(k)} - \sum_{i=1}^{m} w_i\phi_i\left(\mathbf{x}^{(k)}\right)\right]^2$$

$$y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i\phi_i(\mathbf{x})$$



Learn the Optimal Weight Vector

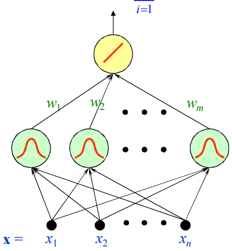
Training set $\mathcal{T} = \left\{ \left(\mathbf{x}^{(k)}, y^{(k)} \right) \right\}_{k=1}^{p}$

Goal $y^{(k)} \approx f\left(\mathbf{x}^{(k)}\right)$ for all k

$$\min E = \frac{1}{2} \sum_{k=1}^{p} \left[y^{(k)} - f\left(\mathbf{x}^{(k)}\right) \right]^2$$

$$= \frac{1}{2} \sum_{k=1}^{p} \left[y^{(k)} - \sum_{i=1}^{m} w_i \phi_i\left(\mathbf{x}^{(k)}\right) \right]^2$$

$$y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$



Learning Algorithm 3

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

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

MLP vs RBFN

Approximation

- MLP : Global network
  - All inputs cause an output
- RBF : Local network
  - Only inputs near a receptive field produce an activation

Comparison between RBF networks and FFNN:

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

- 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*.

- 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.