Neural Networks

Radial Basis Function

Dr. Petr Musilek

Department of Electrical and Computer Engineering
University of Alberta

Fall 2019

Radial Basis Function (RBF) Networks

RBF networks are based directly the theory of function approximation
They posses the following characteristics:

- They are two-layer feed-forward networks.
  - hidden nodes implement a set of radial basis functions
  - output nodes perform linear combination of hidden signals
- The network training is organized into two stages
  - first the parameters of hidden units are determined;
  - then the weights from the hidden to output layer
- The training/learning is very fast
- The networks are very good at interpolation

Exact interpolation

Exact interpolation of a set of n points: each input vector $\boldsymbol{x}(k) = [x_1(k), x_2(k), \ldots, x_m(k)]$ is mapped onto its corresponding target output $t(k)$. The goal is to find a function $f(\boldsymbol{x})$ such that

$$f\big(\boldsymbol{x}(k)\big) = t(k),\ \forall k = 1, 2, \ldots, n$$

RBF approach uses a set of n basis functions, one for each data point, in form $\Phi(\|\boldsymbol{x} - \mu_k\|)$, where $\Phi(\cdot)$ is specific non–linear function. The output mapping is then obtained as a linear combination of the basis functions

$$f(\boldsymbol{x}) = \sum_{k=1}^{n} w_k \Phi(\|\boldsymbol{x} - \mu_k\|)$$

The goal of exact approximation is to find weights w_k such that the function passes through all data points

Weight determination

We can write

$$f(\boldsymbol{x}(l)) = \sum_{k=1}^{n} w_k \Phi\left(\|\boldsymbol{x}(l) - \mu_k\|\right) = t(l)$$

and thus

$$\boldsymbol{w}\boldsymbol{\Phi} = \boldsymbol{t}$$

Standard matrix inversion techniques can be used to obtain the weights

$$\boldsymbol{w} = \boldsymbol{\Phi}^{-1}\boldsymbol{t}$$

With such set of weights, the function $f(\boldsymbol{x})$ represents a continuous surface that passes *exactly* through each data point.

Common RBFs

1)	Gaussian function	$\Phi(v) = \exp(-\frac{v^2}{2\sigma^2})$
2)	Inverse mq function	$\Phi(v) = (v^2 + \sigma^2)^{-\frac{1}{2}}$
3)	Generalized inverse mq function	$\Phi(v) = (v^2 + \sigma^2)^{-\beta}$
4)	Multi–quadratic function	$\Phi(v) = \sqrt{v^2 + \sigma^2}$
5)	Generalized mq function	$\Phi(v) = (v^2 + \sigma^2)^{\beta}$
6)	Thin plate spline function	$\Phi(v) = v^2 \ln(v)$
7)	Cubic function	$\Phi(v) = v^3$
8)	Linear function	$\Phi(v) = v$
	mq $\sim$ multi-quadratic	$\sigma, \alpha > 0;\ 0 < \beta < 1$

Nice list, but only FYI

It has been shown that properties of the interpolating function are relatively insensitive to the precise form of the basis functions $\Phi(v)$.

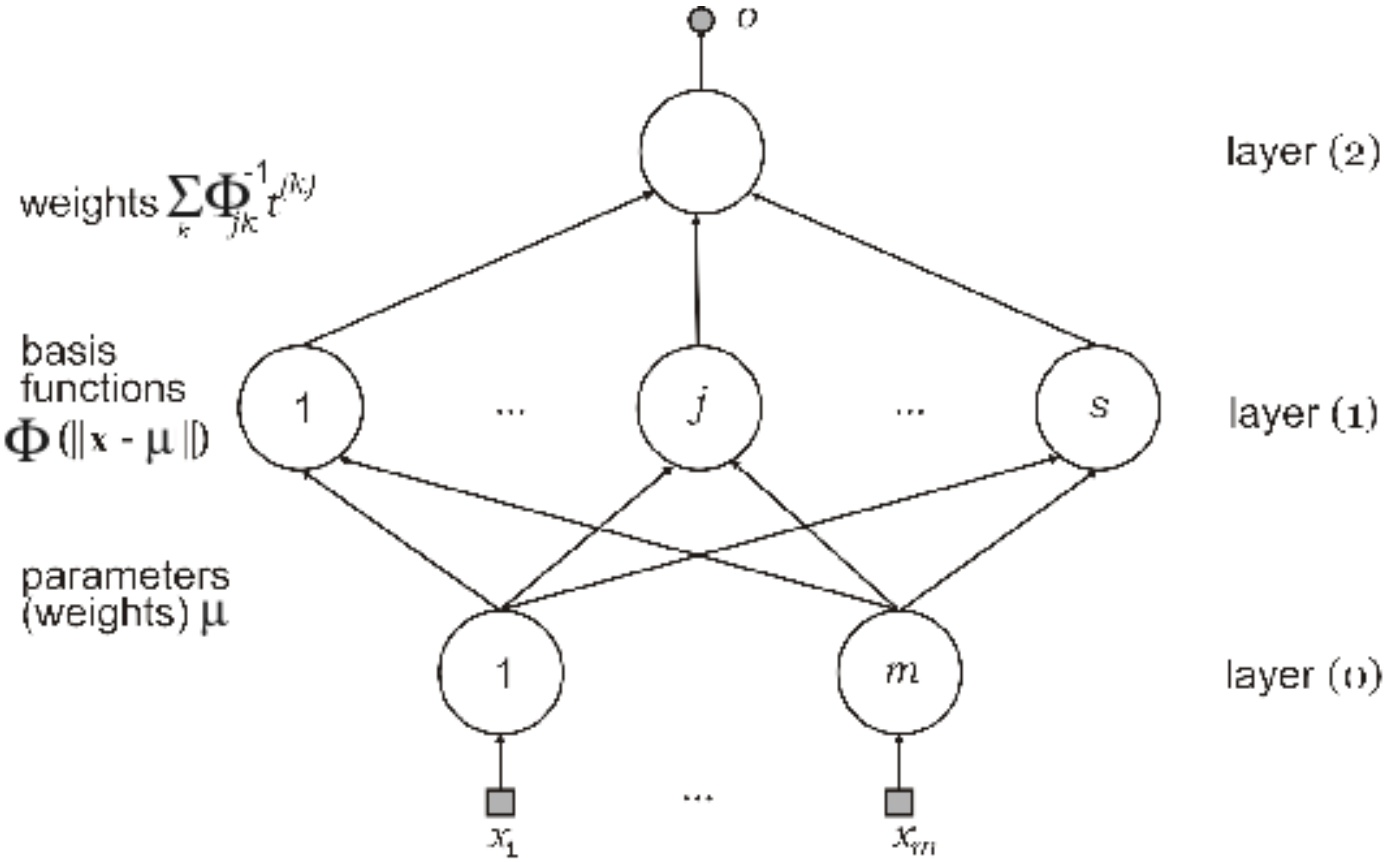
Properties of RBFs

Some (1–3) are localized: $\Phi(v) \to 0$ as $v \to \infty$,
while others (4–8) are not: $\Phi(v) \to \infty$ as $v \to \infty$

All are non-linear in terms of inputs (including the linear function $\Phi(v) = v = \|\boldsymbol{x} - \boldsymbol{\mu}\|$).

- For neural network mappings, localized functions are more suitable (due to the linear combinations of hidden signals in the output layer)
- Gaussian basis function is the most commonly used.

Exact interpolation as NN



Weights are determined directly by n training patterns $\{\boldsymbol{x}, t\}$

Shortcomings of Exact Interpolation

- Poor performance with noisy data (the mapping passes through all data points presented)
- Poor computational efficiency (one hidden unit for each training pattern)
- Possible improvements (to get “real” RBF networks):
  - There can be more than one output neuron, $q > 1$
  - Number of hidden units s could be less then the number of training patterns n (and likely $s \ll n$)
  - Centres of BF could be determined by learning (not set by input data vectors)
  - BF can have variable width (a.k.a. spread σ)
  - Bias parameter could be introduced in the output layer to allow correct mappings without extreme weights

RBF Network

A more general (and powerful) RBF can be described as follows:

$$o_l(\boldsymbol{x}) = w_{l0} + \sum_{j=1}^{s} w_{jl}\Phi_j(\boldsymbol{x})$$

or, by introducing an extra BF $\Phi_0 = 1$

$$o_l(\boldsymbol{x}) = \sum_{j=0}^{s} w_{jl}\Phi_j(\boldsymbol{x})$$

The remaining BFs are usually in the form of Gaussian

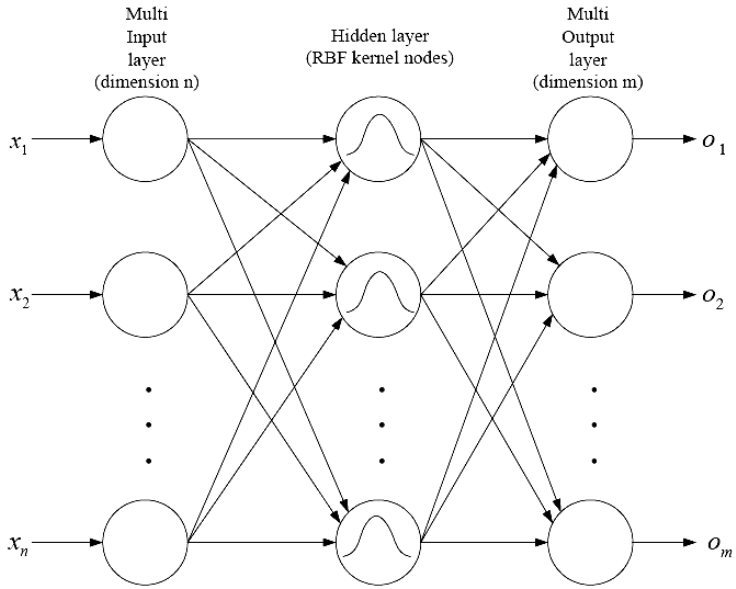
$$\Phi_j(\boldsymbol{x}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right)$$

Network parameters

How to determine the values of parameters w_{kj}, $\boldsymbol{\mu}_j$, and σ_j?

RBF Topology

- Feedforward networks with
  - unity input weights,
  - single hidden layer,
  - specialized activation functions (RBF), and
  - output layer performing linear combination of hidden values



Performance of RBF Networks

It has been shown that linear superposition of localized basis functions provide *universal approximation* capability

Training RBF networks

The proof about universal approximation capabilities of RBF networks are existential (as in the case of MLP) and they do not propose any mechanism how to find structure and parameters of such networks. Unlike in MLP:

- the weights μ_j represent points in m–dimensional input space to which network inputs are compared using a distance measure
- the weights w_{jl} provide coefficients weighting individual BF in their linear combination

RBF is a two–stage learning process that uses algorithms different from those used in MLPs

BF optimization

- The first stage of the learning process
- Designed to set the parameters of BF

- There are several possible approaches
  1. Random selection of fixed centres
  2. Orthogonal least squares
  3. K–means clustering

- They are all unsupervised techniques
- Remaining problem: determining appropriate value of s (the number of BFs)

BF optimization: random selection

Simplest (and quickest) approach

- BF centres fixed at $s < n$ points selected at random from n training data points;

$$\Phi_j \boldsymbol{x} = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right); \text{where } \{\boldsymbol{\mu}_j\} \subset T$$

- BF widths fixed at an appropriate size $\sim$ distribution of training data

$$\sigma_j = \frac{d_{\max}}{\sqrt{2s}}, \text{ or } \sigma_j = 2d_{\text{avg}}$$

where $d_{\max}$ and d_{avg} are maximum and average distance between chosen centers $\boldsymbol{\mu}_j$, respectively.

- Individual BFs are not too wide or narow
- Works well for large training data sets

BF optimization: orthogonal least squares

More advanced approach

- BF are added sequentially, each centered on a data point
- At each step, each potential l–th BF is tested using remaining $n - l$ data points
- BF providing smallest residual output error is selected (although targets for hidden neurons are unknown, the potential change in error can be determined using *projection matrix* which describes relation between desired and observed outputs for optimal, yet unknown, weights)
- To obtain good generalization, cross–validation is used to determine when to stop the addition process

From a set of orthogonal vectors in the space of hidden unit activations, data points for selecting the BF can be calculated directly

BF optimization: K-means clustering

Considers distribution of training data more accurately

- Select K centres at random
- Iteratively re-estimate to partition the training data into K disjoint subsets S_j containing N_{S_j} data points to minimize

$$\text{clustering function} \quad J = \sum_{j=1}^{K} \sum_{k \in S_j} \|\boldsymbol{x}(k) - \boldsymbol{\mu}_j\|^2$$

where $\boldsymbol{\mu}_j$ is the centroid (mean) of the data points in set S_j

$$\boldsymbol{\mu}_j = \frac{1}{N_{S_j}} \sum_{k \in S_j} \boldsymbol{x}(k)$$

- Finally, σ_j are set according to variances of points in individual clusters.

Determining the output weights

$\Phi_j(\boldsymbol{x}, \boldsymbol{\mu}_j, \sigma_j)$ are fixed – we have a single–layer linear network SSE same as for MLP $E = \frac{1}{2}\sum_{k=1}^{n}\sum_{l=1}^{q}[o_l(k) - t_l(k)]^2$ but outputs are simple linear combination of hidden activations

$$o_l(\boldsymbol{x}(k)) = \sum_{j=0}^{s} w_{jl}\Phi_j(\boldsymbol{x}(k))$$

At the minimum of E, the gradients with respect to weights

$$\frac{\partial E}{\partial w_{li}} = \sum_{k=1}^{n}\left(\sum_{j=0}^{s} w_{jl}\Phi_j(\boldsymbol{x}(k)) - t_l(k)\right)\Phi_i(\boldsymbol{x}(k)) = 0$$

which can be solved analytically !

Calculating the output weights

The equation minimizing the gradient can be written in matrix form

$$\boldsymbol{\Phi}^{\mathrm{T}}\left(\boldsymbol{\Phi}\boldsymbol{W}^{\mathrm{T}}-\boldsymbol{T}\right)=0$$

with the formal solution $\boldsymbol{W}^{\mathrm{T}}=\boldsymbol{\Phi}^{+}\boldsymbol{T}$, where $\boldsymbol{\Phi}^{+}$ is pseudo inverse of matrix $\boldsymbol{\Phi}$

$$\boldsymbol{\Phi}^{+}\equiv\left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}$$

which can be solved by fast linear matrix inversion algorithms.

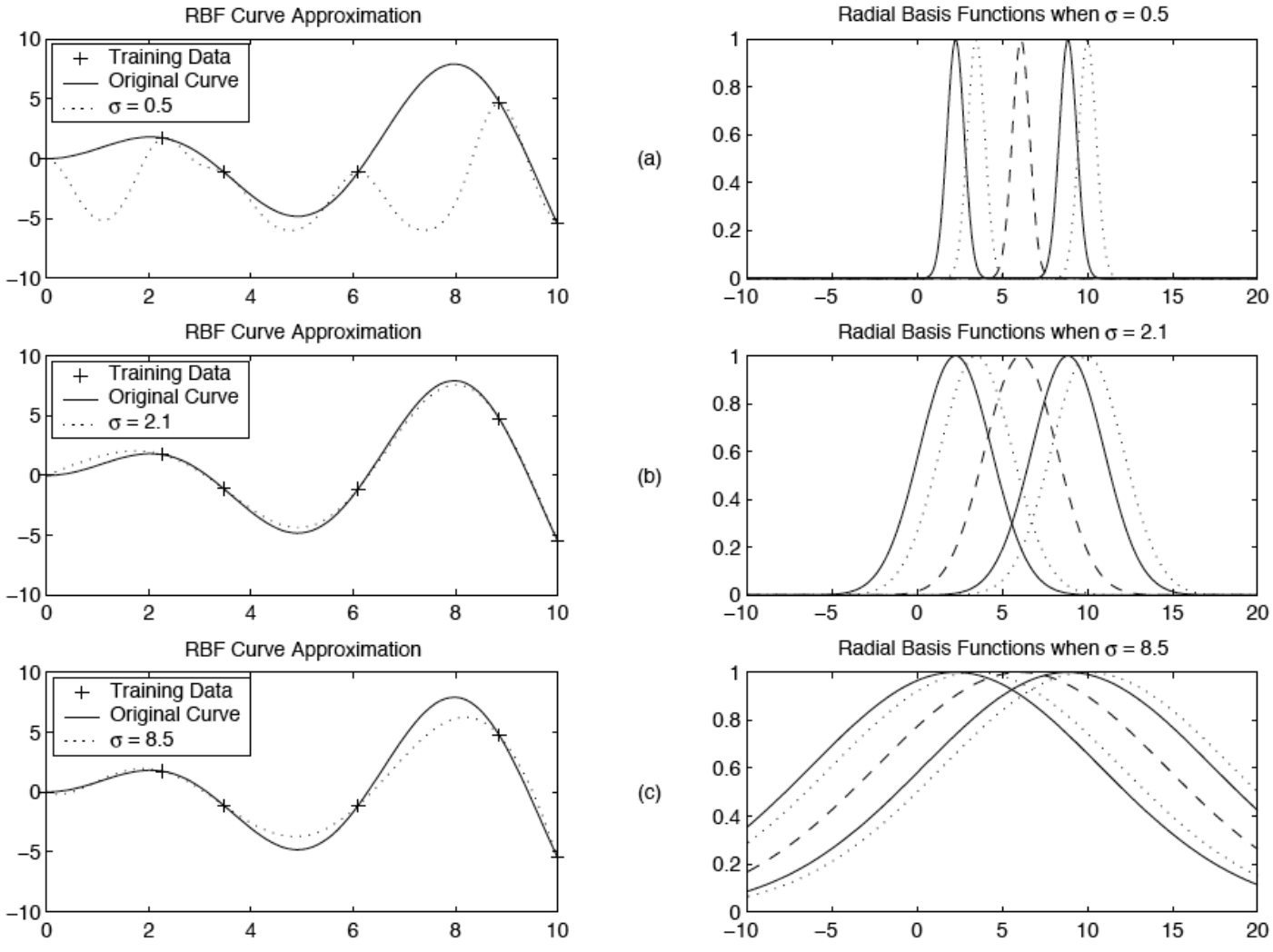
Properties of RBF

- Outputs provide a degree of membership to a class (cluster of data) when used to categorize
- The algorithm does not get stuck in local minima (as is the case with gradient descent learning)
- The first (unsupervised) learning stage is not sensitive to the order of training data

MLP vs. RBF

	MLP	**RBF**
Classification	hyperplanes	hyperspheres
Learning	distributed	localized and faster
Structure	1 or more hidden layers	1 hidden layer but more hidden neurons

Function approximation with RBF Networks



Function approximation with MLP Networks