

Computational Intelligence: Methods and Applications

Lecture 29 Approximation theory, RBF and SFN networks

Włodzisław Duch
SCE, NTU, Singapore
Google: Duch

Basis set functions

A combination of m functions

$$\Phi(\mathbf{X}) = \sum_{i=1}^m w_i \Phi_i(\mathbf{X})$$

may be used for discrimination or for density estimation.
What type of functions are useful here?

Most basis functions are composed of two functions $\Phi(\mathbf{X})=g(f(\mathbf{X}))$:

$f(\mathbf{X})$ activation, defining how to use input features, returning a scalar f .
 $g(f)$ output, converting activation into a new, transformed feature.

Example: multivariate Gaussian function, localized at \mathbf{R} :

$$f(\mathbf{X}) = \|\mathbf{X} - \mathbf{R}\|; \quad g(f) = \exp(-f^2)$$

Activation f . computes distance, output f . localizes it around zero.

Radial functions

General form of multivariate Gaussian:

$$f(\mathbf{X}) = \frac{1}{2}(\mathbf{X} - \mathbf{R})^T \Sigma^{-1}(\mathbf{X} - \mathbf{R}) = \|\mathbf{X} - \mathbf{R}\|_{\Sigma}; \quad g(f) = \exp(-f^2)$$

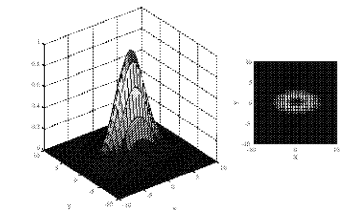
This is a radial basis function (RBF), with Mahalanobis distance and Gaussian decay, a popular choice in neural networks and approximation theory. Radial functions are spherically symmetric in respect to some center. Some examples of RBF functions:

Distance	$f(r) = r = \ \mathbf{X} - \mathbf{R}\ $
Inverse multiquadratic	$h(r) = (\sigma^2 + r^2)^{-\alpha}, \alpha > 0$
Multiquadratic	$h(r) = (\sigma^2 + r^2)^{\beta}, 1 > \beta > 0$
Thin splines	$h(r) = (\sigma r)^2 \ln(\sigma r)$

G + r functions

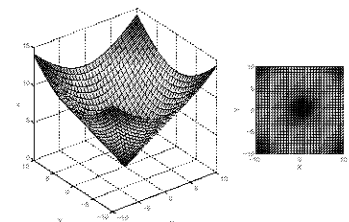
Multivariate Gaussian function
and its contour.

$$G(r) = e^{-r^2}; \quad r = \|\mathbf{X} - \mathbf{R}\|_{\Sigma}$$



Distance function
and its contour.

$$h_d(r) = r = \|\mathbf{X} - \mathbf{R}\|$$

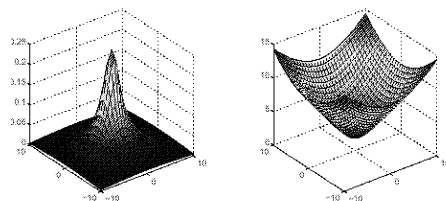


Multiquadratic and thin spline

Multiquadratic and an inverse

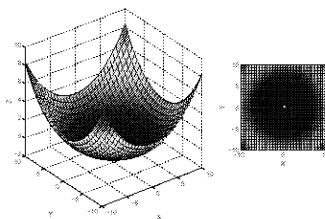
$$h_\alpha(r) = (\sigma^2 + r^2)^{-\alpha}, \quad \alpha = 1;$$

$$h_\beta(r) = (\sigma^2 + r^2)^\beta, \quad \beta = 1/2$$



Thin spline function

$$h_\sigma(r) = (\sigma r)^2 \ln(\sigma r)$$



All these functions are useful in theory of function approximation.

Scalar product activation

Radial functions are useful for density estimation and function approximation. For discrimination, creation of decision borders, activation function equal to linear combination of inputs is most useful:

$$f(\mathbf{X}; \mathbf{W}) = \mathbf{W} \cdot \mathbf{X} = \sum_{i=1}^N W_i X_i$$

Note that this activation may be presented as

$$\mathbf{W} \cdot \mathbf{X} = \frac{1}{2} (\|\mathbf{W}\|^2 + \|\mathbf{X}\|^2 - \|\mathbf{W} - \mathbf{X}\|^2) = L(\mathbf{W}, \mathbf{X}) - D(\mathbf{W}, \mathbf{X})^2$$

The first term L is constant if the length of \mathbf{W} and \mathbf{X} is fixed. This is true for standardized data vectors; square of Euclidean distance is equivalent (up to a constant) to a scalar product!

If $\|\mathbf{X}\|=1$ replace $\mathbf{W} \cdot \mathbf{X}$ by $\|\mathbf{W} - \mathbf{X}\|^2$ and decision borders will still be linear, but using instead of Euclidean various other distance functions will lead to non-linear decision borders!

More basis set functions

More sophisticated combinations of activation functions are useful, ex:

$$f(\mathbf{X}; \mathbf{W}, \mathbf{D}) = \alpha \mathbf{W} \cdot \mathbf{X} + \beta \|\mathbf{X} - \mathbf{D}\|$$

This is a combination of distance-based activation with scalar product activation, allowing to achieve very flexible PDF/decision border shapes. Another interesting choice is separable activation function:

$$f(\mathbf{X}; \theta) = \prod_{i=1}^d f_i(X_i; \theta_i)$$

Separable functions with Gaussian factors have radial form, but Gaussian is the only localized radial function that is also separable.

The $f_i(X; \theta)$ factors may represent probabilities (like in the Naive Bayes method), estimated from histograms using Parzen windows, or may be modeled using some functional form or logical rule.

Output functions

Gaussians and similar "bell shaped" functions are useful to localize output in some region of space.

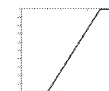
For discrimination weighted combination $f(\mathbf{X}; \mathbf{W}) = \mathbf{W} \cdot \mathbf{X}$ is filtered through a step function, or to create a gradual change through a function with sigmoidal shape (called "squashing f.", such as the logistic function:

$$\sigma(x; \beta) = \frac{1}{1 + \exp(-\beta x)} \in (0, 1)$$

Parameter β sets the slope of the sigmoidal function.

Other commonly used function are:

$\tanh(\beta f) \in (-1, +1)$, similar to logistic function;
semi-linear function: first constant -1 , then linear and then constant $+1$.



Convergence properties

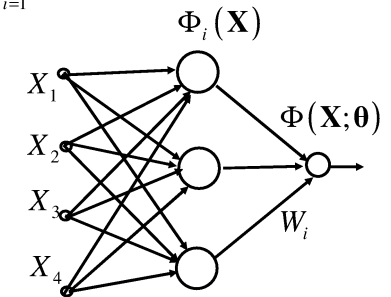
- Multivariate Gaussians and weighted sigmoidal functions may approximate any function: such systems are universal approximators.
- The choice of functions determines the speed of convergence of the approximation and the number of functions need for approximation.
- The approximation error in d -dimensional spaces using weighted activation with sigmoidal functions does not depend on d . The rate of convergence with m functions is $O(1/m)$
- Polynomials, orthogonal polynomials etc need for reliable estimation a number of points that grows exponentially with d like making them useless for high-dimensional problems! $O\left(\frac{1}{n^{1/d}}\right)$
The error convergence rate is:
In 2-D we need 10 time more data points to achieve the same error as in 1D, but in 10-D we need 10G times more points!

Radial basis networks (RBF)

RBF is a linear approximation in space of radial basis functions

$$\Phi(\mathbf{X}; \boldsymbol{\theta}) = \sum_{i=1}^m W_i \Phi(\|\mathbf{X} - \mathbf{X}^{(i)}\|; \theta_i) = \sum_{i=1}^m W_i \Phi_i(\mathbf{X})$$

Such computations are frequently presented in a network form:
inputs nodes: X_i values;
internal (hidden) nodes: functions;
outgoing connections: W_i coefficients.
output node: summation.



Sometimes RBF networks are called "neural", due to inspiration for their development.

RBF for approximation

RBF networks may be used for function approximation, or classification with infinitely many classes. Function should pass through points:

$$\Phi(\mathbf{X}^{(i)}; \boldsymbol{\theta}) = Y^{(i)}, \quad i = 1 \dots n$$

Approximation function should also be smooth to avoid high variance of the model, but not too smooth, to avoid high bias. Taking n identical functions centered at the data vectors:

$$\Phi(\mathbf{X}^{(i)}; \boldsymbol{\theta}) = \sum_{j=1}^n W_j \Phi(\|\mathbf{X}^{(i)} - \mathbf{X}^{(j)}\|) = \sum_{j=1}^n \mathbf{H}_{ij} W_j = Y^{(i)}$$

$$\mathbf{H}\mathbf{W} = \mathbf{Y} \Rightarrow \mathbf{W} = \mathbf{H}^{-1}\mathbf{Y}$$

If matrix \mathbf{H} is not too big and non-singular this will work; in practice many iterative schemes to solve the approximation problem have been devised. For classification $Y^{(i)}=0$ or 1.

Separable Function Networks (SFN)

For knowledge discovery and mixture of Naive Bayes models separable functions are preferred. Each function component

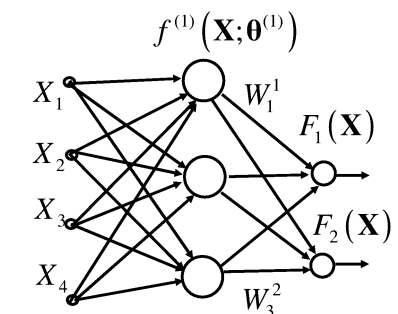
$$f^{(j)}(\mathbf{X}; \boldsymbol{\theta}^{(j)}) = \prod_{i=1}^d f_i(X_i; \theta_i^{(j)})$$

is represented by a single node and if localized functions are used may represent some local conditions.

Linear combination of these component functions:

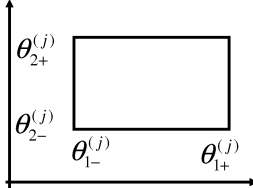
$$F_c(\mathbf{X}; \boldsymbol{\theta}, \mathbf{W}^c) = \sum_{j=1}^{n(c)} W_j^c f^{(j)}(\mathbf{X}; \boldsymbol{\theta}^{(j)})$$

specifies the output; several outputs F_c are defined, for different classes, conclusions, class-conditional probability distributions etc.



SFN for logical rules

If the component functions are rectangular:

$$f_i(X_i; \theta_i^{(j)}) = \begin{cases} 1 & \text{if } X_i \in [\theta_{i-}^{(j)}, \theta_{i+}^{(j)}] \\ 0 & \text{if } X_i \notin [\theta_{i-}^{(j)}, \theta_{i+}^{(j)}] \end{cases}$$


then the product function realized by the node is a hyperrectangle, and it may represent crisp logic rule:

$$\text{IF } X_1 \in [\theta_{1-}^{(j)}, \theta_{1+}^{(j)}] \wedge \dots \wedge X_i \in [\theta_{i-}^{(j)}, \theta_{i+}^{(j)}] \wedge \dots \wedge X_d \in [\theta_{d-}^{(j)}, \theta_{d+}^{(j)}]$$

THEN Fact^(j) = True

Conditions that cover whole data may be deleted.

SNF rules

Final function is a sum of all rules for a given fact (class).

$$F_c(\mathbf{X}; \boldsymbol{\theta}, \mathbf{W}^c) = \sum_{j=1}^{n(c)} W_j^c f^{(j)}(\mathbf{X}; \boldsymbol{\theta}^{(j)})$$

The output weights are either:

all $W_j = 1$, all rules are on equal footing;

$W_j \sim$ Rule precision (confidence): a ratio of the number of vectors correctly covered by the rule over the number of all elements covered.

$$W_j = N(\mathbf{X} \in \text{CI Rule}_j) / N(\mathbf{X} \in \text{Rule}_j)$$

This may additionally be multiplied by the coverage of the rule, or

$$W_j = N(\mathbf{X} \in \text{CI Rule}_j)^2 / N(\mathbf{X} \in \text{Rule}_j) N(\mathbf{X} \in C)$$

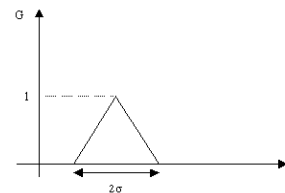
W may also be fitted to data to increase accuracy of predictions.

Rules with weighted conditions

Instead of rectangular functions Gaussian, triangular or trapezoidal functions may be used to evaluate the degree (not always equivalent to probability) of a condition being fulfilled.

For example, triangular functions

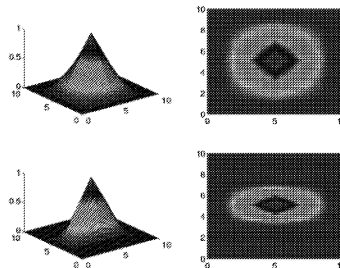
$$f(X_i; D_i, \sigma_i) = \max\left(0, 1 - \frac{|X_i - D_i|}{\sigma_i}\right)$$



A fuzzy rule based on "triangular membership functions" is a product of such functions (conditions):

$$f(\mathbf{X}; \mathbf{D}, \boldsymbol{\sigma}) = \prod_{i=1}^d f_i(X_i; D_i, \sigma_i)$$

The conclusion is highly justified in areas where $f()$ is large, shapes =>



RBFs and SFNs

Many basis set expansions have been proposed in approximation theory.

In some branches of science and engineering such expansions have been widely used, for example in computational chemistry.

There is no particular reason why radial functions should be used, but all basis set expansions are mistakenly called now RBFs ...

In practice Gaussian functions are used most often, and Gaussian approximators and classifiers have been used long before RBFs.

Gaussian functions are also separable, so RBF=SFN for Gaussians.

For other functions:

- SFNs have natural interpretation in terms of fuzzy logic membership functions and trains as neurofuzzy systems.
- SFNs can be used to extract logical (crisp and fuzzy) rules from data.
- SFNs may be treated as extension of Naive Bayes, with voting committees of NB models.
- SFNs may be used in combinatorial reasoning (see Lecture 31).

but remember ...

that all this is just a poor approximation to Bayesian analysis.

It allows to model situations where we have linguistic knowledge but no data – for Bayesians one may say that we guess prior distributions from rough descriptions and improve the results later by collecting real data.

Example: [RBF regression](#)

Neural Java tutorial: <http://diwww.epfl.ch/mantra/tutorial/>

[Transfer function interactive tutorial.](#)