



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages 
Español
Italiano
Polski
Русский
中文


 Edit links

Create account Log in

Article [Talk](#)

[Read](#) [Edit](#)

[More](#) ▾



Radial basis function network

From Wikipedia, the free encyclopedia

In the field of [mathematical modeling](#), a **radial basis function network** is an [artificial neural network](#) that uses [radial basis functions](#) as [activation functions](#). The output of the network is a [linear combination](#) of radial basis functions of the inputs and neuron parameters. Radial basis function networks have many uses, including [function approximation](#), [time series prediction](#), [classification](#), and system [control](#). They were first formulated in a 1988 paper by Broomhead and Lowe, both researchers at the [Royal Signals and Radar Establishment](#).^{[1][2][3]}

Contents [hide]

- 1 Network architecture
 - 1.1 Normalized
 - 1.1.1 Normalized architecture
 - 1.1.2 Theoretical motivation for normalization
 - 1.2 Local linear models
- 2 Training
 - 2.1 Interpolation
 - 2.2 Function approximation
 - 2.2.1 Training the basis function centers
 - 2.2.2 Pseudoinverse solution for the linear weights
 - 2.2.3 Gradient descent training of the linear weights
 - 2.2.4 Projection operator training of the linear weights
- 3 Examples
 - 3.1 Logistic map
 - 3.2 Function approximation
 - 3.2.1 Unnormalized radial basis functions
 - 3.2.2 Normalized radial basis functions
 - 3.3 Time series prediction
 - 3.4 Control of a chaotic time series
- 4 See also
- 5 References

Network architecture [\[edit\]](#)

Radial basis function (RBF) networks typically have three layers: an input layer, a hidden layer with a non-linear RBF activation function and a linear output layer. The input can be modeled as a vector of real numbers $\mathbf{x} \in \mathbb{R}^n$. The output of the network is then a scalar function of the input vector, $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}$, and is given by

$$\varphi(\mathbf{x}) = \sum_{i=1}^N a_i \rho(\|\mathbf{x} - \mathbf{c}_i\|)$$

where N is the number of neurons in the hidden layer, \mathbf{c}_i is the center vector for neuron i , and a_i is the weight of neuron i in the linear output neuron. Functions that depend only on the distance from a center vector are radially symmetric about that vector, hence the name radial basis function. In the basic form all inputs are connected to each hidden neuron. The [norm](#) is typically taken to be the [Euclidean distance](#) (although the [Mahalanobis distance](#) appears to perform better in general^[*citation needed*]) and the radial basis function is commonly taken to be [Gaussian](#)

$$\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp[-\beta \|\mathbf{x} - \mathbf{c}_i\|^2].$$

The Gaussian basis functions are local to the center vector in the sense that

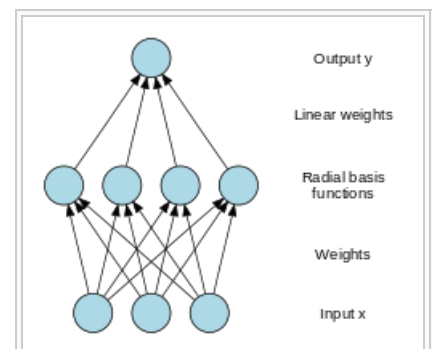


Figure 1: Architecture of a radial basis function network. An input vector \mathbf{x} is used as input to all radial basis functions, each with different parameters. The output of the network is a linear combination of the outputs from radial basis functions.

$$\lim_{\|x\| \rightarrow \infty} \rho(\|x - c_i\|) = 0$$

i.e. changing parameters of one neuron has only a small effect for input values that are far away from the center of that neuron.

Given certain mild conditions on the shape of the activation function, RBF networks are [universal approximators](#) on a [compact](#) subset of \mathbb{R}^n .^[4] This means that an RBF network with enough hidden neurons can approximate any continuous function with arbitrary precision.

The parameters a_i , c_i , and β_i are determined in a manner that optimizes the fit between φ and the data.

Normalized [\[edit\]](#)

Normalized architecture [\[edit\]](#)

In addition to the above *unnormalized* architecture, RBF networks can be *normalized*. In this case the mapping is

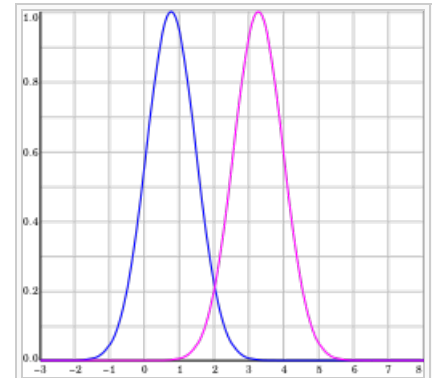


Figure 2: Two unnormalized radial basis functions in one input dimension. The basis function centers are located at $c_1 = 0.75$ and $c_2 = 3.25$.

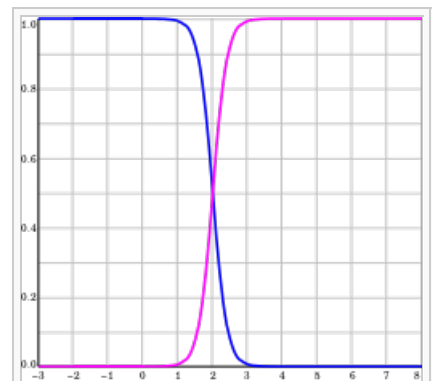


Figure 3: Two normalized radial basis functions in one input dimension. The basis function centers are located at $c_1 = 0.75$ and $c_2 = 3.25$.



Figure 4: Three normalized radial basis functions in one input dimension. The additional basis function has center at $c_3 = 2.75$

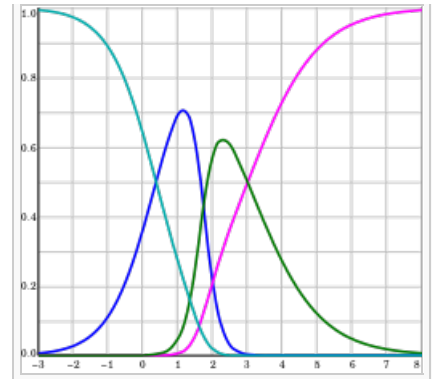


Figure 5: Four normalized radial basis functions in one input dimension. The fourth basis function has center at $\mathbf{c}_4 = 0$. Note that the first basis function (dark blue) has become localized.

$$\varphi(\mathbf{x}) \stackrel{\text{def}}{=} \frac{\sum_{i=1}^N a_i \rho(\|\mathbf{x} - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|)} = \sum_{i=1}^N a_i u(\|\mathbf{x} - \mathbf{c}_i\|)$$

where

$$u(\|\mathbf{x} - \mathbf{c}_i\|) \stackrel{\text{def}}{=} \frac{\rho(\|\mathbf{x} - \mathbf{c}_i\|)}{\sum_{j=1}^N \rho(\|\mathbf{x} - \mathbf{c}_j\|)}$$

is known as a "normalized radial basis function".

Theoretical motivation for normalization [\[edit\]](#)

There is theoretical justification for this architecture in the case of stochastic data flow. Assume a [stochastic kernel](#) approximation for the joint probability density

$$P(\mathbf{x} \wedge y) = \frac{1}{N} \sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|) \sigma(|y - e_i|)$$

where the weights \mathbf{c}_i and e_i are exemplars from the data and we require the kernels to be normalized

$$\int \rho(\|\mathbf{x} - \mathbf{c}_i\|) d^n \mathbf{x} = 1$$

and

$$\int \sigma(|y - e_i|) dy = 1.$$

The probability densities in the input and output spaces are

$$P(\mathbf{x}) = \int P(\mathbf{x} \wedge y) dy = \frac{1}{N} \sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|)$$

and

The expectation of y given an input \mathbf{x} is

$$\varphi(\mathbf{x}) \stackrel{\text{def}}{=} E(y | \mathbf{x}) = \int y P(y | \mathbf{x}) dy$$

where

$$P(y | \mathbf{x})$$

is the conditional probability of y given \mathbf{x} . The conditional probability is related to the joint probability through [Bayes theorem](#)

$$P(y | \mathbf{x}) = \frac{P(\mathbf{x} \wedge y)}{P(\mathbf{x})}$$

which yields

$$\varphi(\mathbf{x}) = \int y \frac{P(\mathbf{x} \wedge y)}{P(\mathbf{x})} dy.$$

This becomes

$$\varphi(\mathbf{x}) = \frac{\sum_{i=1}^N e_i \rho(\|\mathbf{x} - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|)} = \sum_{i=1}^N e_i u(\|\mathbf{x} - \mathbf{c}_i\|)$$

when the integrations are performed.

Local linear models [\[edit\]](#)

It is sometimes convenient to expand the architecture to include [local linear](#) models. In that case the architectures become, to first order,

$$\varphi(\mathbf{x}) = \sum_{i=1}^N (a_i + \mathbf{b}_i \cdot (\mathbf{x} - \mathbf{c}_i)) \rho(\|\mathbf{x} - \mathbf{c}_i\|)$$

and

$$\varphi(\mathbf{x}) = \sum_{i=1}^N (a_i + \mathbf{b}_i \cdot (\mathbf{x} - \mathbf{c}_i)) u(\|\mathbf{x} - \mathbf{c}_i\|)$$

in the unnormalized and normalized cases, respectively. Here \mathbf{b}_i are weights to be determined. Higher order linear terms are also possible.

This result can be written

$$\varphi(\mathbf{x}) = \sum_{i=1}^{2N} \sum_{j=1}^n e_{ij} v_{ij}(\mathbf{x} - \mathbf{c}_i)$$

where

$$e_{ij} = \begin{cases} a_i, & \text{if } i \in [1, N] \\ b_{ij}, & \text{if } i \in [N + 1, 2N] \end{cases}$$

and

$$v_{ij}(\mathbf{x} - \mathbf{c}_i) \stackrel{\text{def}}{=} \begin{cases} \delta_{ij} \rho(\|\mathbf{x} - \mathbf{c}_i\|), & \text{if } i \in [1, N] \\ (x_{ij} - c_{ij}) \rho(\|\mathbf{x} - \mathbf{c}_i\|), & \text{if } i \in [N + 1, 2N] \end{cases}$$

in the unnormalized case and

$$v_{ij}(\mathbf{x} - \mathbf{c}_i) \stackrel{\text{def}}{=} \begin{cases} \delta_{ij} u(\|\mathbf{x} - \mathbf{c}_i\|), & \text{if } i \in [1, N] \\ (x_{ij} - c_{ij}) u(\|\mathbf{x} - \mathbf{c}_i\|), & \text{if } i \in [N + 1, 2N] \end{cases}$$

in the normalized case.

Here δ_{ij} is a [Kronecker delta function](#) defined as

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

Training [\[edit\]](#)

RBF networks are typically trained by a two-step algorithm. In the first step, the center vectors \mathbf{c}_i of the RBF functions in the hidden layer are chosen. This step can be performed in several ways; centers can be randomly sampled from some set of examples, or they can be determined using [k-means clustering](#). Note that this step is [unsupervised](#). A third [backpropagation](#) step can be performed to fine-tune all of the RBF net's parameters.^[3]

The second step simply fits a linear model with coefficients \mathbf{w}_i to the hidden layer's outputs with respect to some objective function. A common objective function, at least for regression/function estimation, is the least squares function:

$$K(\mathbf{w}) \stackrel{\text{def}}{=} \sum_{t=1}^{\infty} K_t(\mathbf{w})$$

where

$$K_t(\mathbf{w}) \stackrel{\text{def}}{=} [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})]^2.$$

We have explicitly included the dependence on the weights. Minimization of the least squares objective function

by optimal choice of weights optimizes accuracy of fit.

There are occasions in which multiple objectives, such as smoothness as well as accuracy, must be optimized. In that case it is useful to optimize a regularized objective function such as

$$H(\mathbf{w}) \stackrel{\text{def}}{=} K(\mathbf{w}) + \lambda S(\mathbf{w}) \stackrel{\text{def}}{=} \sum_{t=1}^{\infty} H_t(\mathbf{w})$$

where

$$S(\mathbf{w}) \stackrel{\text{def}}{=} \sum_{t=1}^{\infty} S_t(\mathbf{w})$$

and

$$H_t(\mathbf{w}) \stackrel{\text{def}}{=} K_t(\mathbf{w}) + \lambda S_t(\mathbf{w})$$

where optimization of S maximizes smoothness and λ is known as a [regularization](#) parameter.

Interpolation [\[edit\]](#)

RBF networks can be used to interpolate a function $y: \mathbb{R}^n \rightarrow \mathbb{R}$ when the values of that function are known on finite number of points: $y(\mathbf{x}_i) = b_i, i = 1, \dots, N$. Taking the known points \mathbf{x}_i to be the centers of the radial basis functions and evaluating the values of the basis functions at the same points

$g_{ij} = \rho(\|\mathbf{x}_j - \mathbf{x}_i\|)$ the weights can be solved from the equation

$$\begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1N} \\ g_{21} & g_{22} & \cdots & g_{2N} \\ \vdots & & \ddots & \vdots \\ g_{N1} & g_{N2} & \cdots & g_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$$

It can be shown that the interpolation matrix in the above equation is non-singular, if the points \mathbf{x}_i are distinct, and thus the weights \mathbf{w} can be solved by simple linear algebra:

$$\mathbf{w} = \mathbf{G}^{-1}\mathbf{b}$$

Function approximation [\[edit\]](#)

If the purpose is not to perform strict interpolation but instead more general [function approximation](#) or [classification](#) the optimization is somewhat more complex because there is no obvious choice for the centers. The training is typically done in two phases first fixing the width and centers and then the weights. This can be justified by considering the different nature of the non-linear hidden neurons versus the linear output neuron.

Training the basis function centers [\[edit\]](#)

Basis function centers can be randomly sampled among the input instances or obtained by Orthogonal Least Square Learning Algorithm or found by [clustering](#) the samples and choosing the cluster means as the centers.

The RBF widths are usually all fixed to same value which is proportional to the maximum distance between the chosen centers.

Pseudoinverse solution for the linear weights [\[edit\]](#)

After the centers \mathbf{c}_i have been fixed, the weights that minimize the error at the output are computed with a linear [pseudoinverse](#) solution:

$$\mathbf{w} = \mathbf{G}^+\mathbf{b}$$

where the entries of G are the values of the radial basis functions evaluated at the points \mathbf{x}_i :

$$g_{ji} = \rho(\|\mathbf{x}_j - \mathbf{c}_i\|)$$

The existence of this linear solution means that unlike multi-layer perceptron (MLP) networks, RBF networks have a unique local minimum (when the centers are fixed).

Gradient descent training of the linear weights [\[edit\]](#)

Another possible training algorithm is [gradient descent](#). In gradient descent training, the weights are adjusted at each time step by moving them in a direction opposite from the gradient of the objective function (thus allowing the minimum of the objective function to be found),

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \nu \frac{d}{d\mathbf{w}} H_t(\mathbf{w})$$

where ν is a "learning parameter."

For the case of training the linear weights, \mathbf{a}_i , the algorithm becomes

$$\mathbf{a}_i(t+1) = \mathbf{a}_i(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] \rho(\|\mathbf{x}(t) - \mathbf{c}_i\|)$$

in the unnormalized case and

$$\mathbf{a}_i(t+1) = \mathbf{a}_i(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] u(\|\mathbf{x}(t) - \mathbf{c}_i\|)$$

in the normalized case.

For local-linear-architectures gradient-descent training is

$$\mathbf{e}_{ij}(t+1) = \mathbf{e}_{ij}(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] v_{ij}(\mathbf{x}(t) - \mathbf{c}_i)$$

Projection operator training of the linear weights [\[edit\]](#)

For the case of training the linear weights, \mathbf{a}_i and \mathbf{e}_{ij} , the algorithm becomes

$$\mathbf{a}_i(t+1) = \mathbf{a}_i(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] \frac{\rho(\|\mathbf{x}(t) - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho^2(\|\mathbf{x}(t) - \mathbf{c}_i\|)}$$

in the unnormalized case and

$$\mathbf{a}_i(t+1) = \mathbf{a}_i(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] \frac{u(\|\mathbf{x}(t) - \mathbf{c}_i\|)}{\sum_{i=1}^N u^2(\|\mathbf{x}(t) - \mathbf{c}_i\|)}$$

in the normalized case and

$$\mathbf{e}_{ij}(t+1) = \mathbf{e}_{ij}(t) + \nu [y(t) - \varphi(\mathbf{x}(t), \mathbf{w})] \frac{v_{ij}(\mathbf{x}(t) - \mathbf{c}_i)}{\sum_{i=1}^N \sum_{j=1}^n v_{ij}^2(\mathbf{x}(t) - \mathbf{c}_i)}$$

in the local-linear case.

For one basis function, projection operator training reduces to [Newton's method](#).

Examples [\[edit\]](#)

Logistic map [\[edit\]](#)

The basic properties of radial basis functions can be illustrated with a simple mathematical map, the [logistic map](#), which maps the unit interval onto itself. It can be used to generate a convenient prototype data stream. The logistic map can be used to explore [function approximation](#), [time series prediction](#), and [control theory](#). The map originated from the field of [population dynamics](#) and became the prototype for [chaotic](#) time series. The map, in the fully chaotic regime, is given by

$$x(t+1) \stackrel{\text{def}}{=} f[x(t)] = 4x(t)[1-x(t)]$$

where t is a time index. The value of x at time $t+1$ is a parabolic function of x at time t . This equation represents the underlying geometry of the chaotic time series generated by the logistic map.

Generation of the time series from this equation is the **forward problem**. The examples here illustrate the **inverse problem**; identification of the underlying dynamics, or fundamental equation, of the logistic map from exemplars of the time series. The goal is to find an estimate

$$x(t+1) = f[x(t)] \approx \varphi(t) = \varphi[x(t)]$$

for f .

Function approximation [\[edit\]](#)

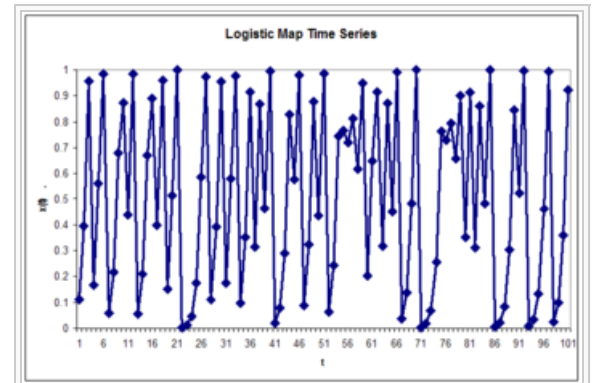


Figure 6: Logistic map time series. Repeated iteration of the [□] logistic map generates a chaotic time series. The values lie between zero and one. Displayed here are the 100 training points used to train the examples in this section. The weights \mathbf{c} are the first five points from this time series.

Unnormalized radial basis functions [\[edit\]](#)

The architecture is

$$\varphi(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{i=1}^N a_i \rho(\|\mathbf{x} - \mathbf{c}_i\|)$$

where

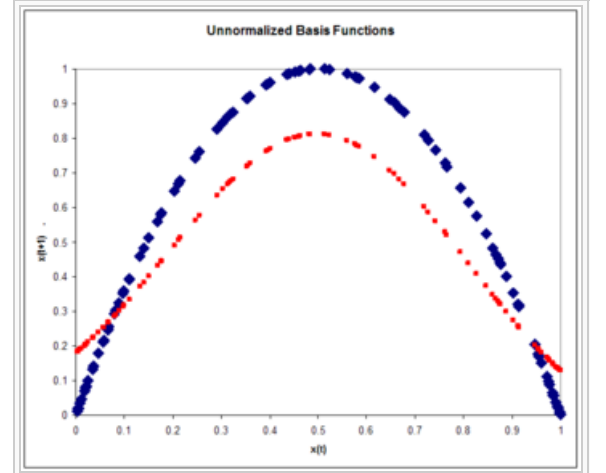


Figure 7: Unnormalized basis functions. The Logistic map (blue) and the approximation to the logistic map (red) after one pass through the training set.

$$\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp[-\beta \|\mathbf{x} - \mathbf{c}_i\|^2] = \exp[-\beta (x(t) - c_i)^2].$$

Since the input is a [scalar](#) rather than a [vector](#), the input dimension is one. We choose the number of basis functions as $N=5$ and the size of the training set to be 100 exemplars generated by the chaotic time series. The weight β is taken to be a constant equal to 5. The weights \mathbf{c}_i are five exemplars from the time series. The weights \mathbf{a}_i are trained with projection operator training:

$$a_i(t+1) = a_i(t) + \nu [x(t+1) - \varphi(\mathbf{x}(t), \mathbf{w})] \frac{\rho(\|\mathbf{x}(t) - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho^2(\|\mathbf{x}(t) - \mathbf{c}_i\|)}$$

where the learning rate ν is taken to be 0.3. The training is performed with one pass through the 100 training points. The [rms error](#) is 0.15.

Normalized radial basis functions [\[edit\]](#)

The normalized RBF architecture is

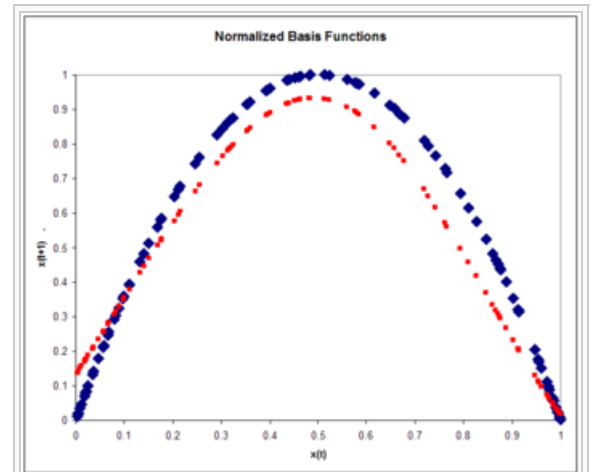


Figure 8: Normalized basis functions. The Logistic map (blue) and the approximation to the logistic map (red) after one pass through the training set. Note the improvement over the unnormalized case.

$$\varphi(\mathbf{x}) \stackrel{\text{def}}{=} \frac{\sum_{i=1}^N a_i \rho(\|\mathbf{x} - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|)} = \sum_{i=1}^N a_i u(\|\mathbf{x} - \mathbf{c}_i\|)$$

where

$$u(\|\mathbf{x} - \mathbf{c}_i\|) \stackrel{\text{def}}{=} \frac{\rho(\|\mathbf{x} - \mathbf{c}_i\|)}{\sum_{i=1}^N \rho(\|\mathbf{x} - \mathbf{c}_i\|)}.$$

Again:

$$\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp[-\beta \|\mathbf{x} - \mathbf{c}_i\|^2] = \exp[-\beta (x(t) - c_i)^2].$$

Again, we choose the number of basis functions as five and the size of the training set to be 100 exemplars generated by the chaotic time series. The weight β is taken to be a constant equal to 6. The weights c_i are five exemplars from the time series. The weights a_i are trained with projection operator training:

$$a_i(t+1) = a_i(t) + \nu [x(t+1) - \varphi(\mathbf{x}(t), \mathbf{w})] \frac{u(\|\mathbf{x}(t) - \mathbf{c}_i\|)}{\sum_{i=1}^N u^2(\|\mathbf{x}(t) - \mathbf{c}_i\|)}$$

where the learning rate ν is again taken to be 0.3. The training is performed with one pass through the 100 training points. The **rms error** on a test set of 100 exemplars is 0.084, smaller than the unnormalized error. Normalization yields accuracy improvement. Typically accuracy with normalized basis functions increases even more over unnormalized functions as input dimensionality increases.

Time series prediction [\[edit\]](#)

Once the underlying geometry of the time series is estimated as in the previous examples, a prediction for the time series can be made by iteration:

$$\begin{aligned}\varphi(0) &= x(1) \\ x(t) &\approx \varphi(t-1) \\ x(t+1) &\approx \varphi(t) = \varphi[\varphi(t-1)].\end{aligned}$$

A comparison of the actual and estimated time series is displayed in the figure. The estimated times series starts out at time zero with an exact knowledge of $x(0)$. It then uses the estimate of the dynamics to update the time series estimate for several time steps.

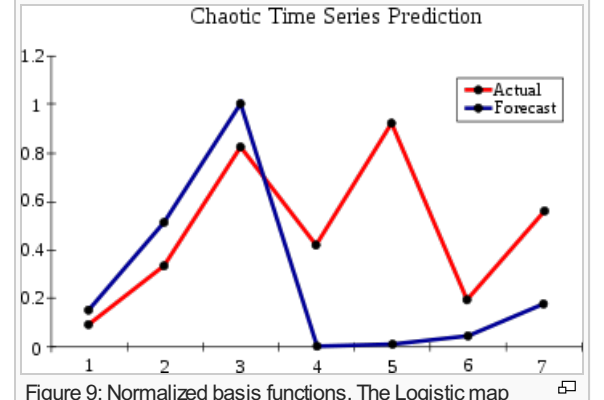


Figure 9: Normalized basis functions. The Logistic map (blue) and the approximation to the logistic map (red) as a function of time. Note that the approximation is good for only a few time steps. This is a general characteristic of chaotic time series.

Note that the estimate is accurate for only a few time steps. This is a general characteristic of chaotic time series. This is a property of the sensitive dependence on initial conditions common to chaotic time series. A small initial error is amplified with time. A measure of the divergence of time series with nearly identical initial conditions is known as the **Lyapunov exponent**.

Control of a chaotic time series [\[edit\]](#)

We assume the output of the logistic map can be manipulated through a control parameter $c[x(t), t]$ such that

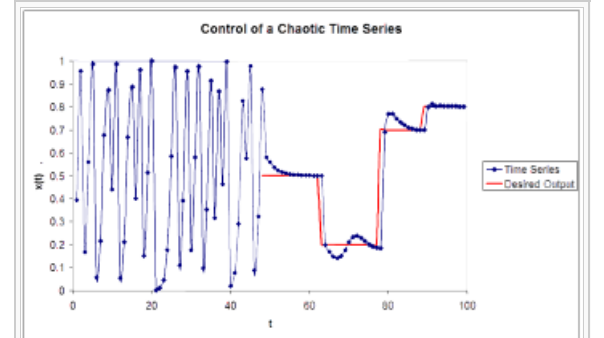


Figure 10: Control of the logistic map. The system is allowed to evolve naturally for 49 time steps. At time 50 control is turned on. The desired trajectory for the time series is red. The system under control learns the underlying dynamics and drives the time series to the desired output. The architecture is the same as for the time series prediction example.

$$x(t+1) = 4x(t)[1-x(t)] + c[x(t), t].$$

The goal is to choose the control parameter in such a way as to drive the time series to a desired output $d(t)$. This can be done if we choose the control parameter to be

$$c[x(t), t] \stackrel{\text{def}}{=} -\varphi[x(t)] + d(t+1)$$

where

$$y[x(t)] \approx f[x(t)] = x(t+1) - c[x(t), t]$$

is an approximation to the underlying natural dynamics of the system.

The learning algorithm is given by

$$a_i(t+1) = a_i(t) + \nu \varepsilon \frac{u(\|\mathbf{x}(t) - \mathbf{c}_i\|)}{\sum_{i=1}^N u^2(\|\mathbf{x}(t) - \mathbf{c}_i\|)}$$

where

$$\varepsilon \stackrel{\text{def}}{=} f[x(t)] - \varphi[x(t)] = x(t+1) - c[x(t), t] - \varphi[x(t)] = x(t+1) - d(t+1)$$

See also [\[edit\]](#)

- [Radial basis function kernel](#)
- [In Situ Adaptive Tabulation](#)
- [Predictive analytics](#)
- [Chaos theory](#)

References [\[edit\]](#)

- [↑] Broomhead, D. S.; Lowe, David (1988). *Radial basis functions, multi-variable functional interpolation and adaptive networks* [\[](#)Technical report]. *RSRE*. 4148.
- [↑] Broomhead, D. S.; Lowe, David (1988). "Multivariable functional interpolation and adaptive networks". *Complex Systems* **2**: 321–355.
- [↑] ^{*a*} ^{*b*} Schwenker, Friedhelm; Kestler, Hans A.; Palm, Günther (2001). "Three learning phases for radial-basis-function networks". *Neural Networks* **14**: 439–458. doi:10.1016/s0893-6080(01)00027-2 [\[](#)CiteSeerX 10.1.1.109.312].
- [↑] Park, J.; I. W. Sandberg (Summer 1991). "Universal Approximation Using Radial-Basis-Function Networks" [\[](#)]. *Neural Computation* **3** (2): 246–257. doi:10.1162/neco.1991.3.2.246 [\[](#)]. Retrieved 26 March 2013.
- J. Moody and C. J. Darken, "Fast learning in networks of locally tuned processing units," *Neural Computation*, 1, 281-294 (1989). Also see [Radial basis function networks according to Moody and Darken](#) [\[](#)]
- T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE* 78(9), 1484-1487 (1990).
- [Roger D. Jones](#), Y. C. Lee, C. W. Barnes, G. W. Flake, K. Lee, P. S. Lewis, and S. Qian, ["Function approximation and time series prediction with neural networks"](#) [\[](#)],? *Proceedings of the International Joint Conference on Neural Networks*, June 17–21, p. I-649 (1990).
- Martin D. Buhmann (2003). *Radial Basis Functions: Theory and Implementations*. Cambridge University. ISBN 0-521-63338-9.
- Yee, Paul V. and Haykin, Simon (2001). *Regularized Radial Basis Function Networks: Theory and Applications*. John Wiley. ISBN 0-471-35349-3.
- John R. Davies, Stephen V. Coggeshall, [Roger D. Jones](#), and Daniel Schutzer, "Intelligent Security Systems," in Freedman, Roy S., Flein, Robert A., and Lederman, Jess, Editors (1995). *Artificial Intelligence in the Capital Markets*. Chicago: Irwin. ISBN 1-55738-811-3.
- Simon Haykin (1999). *Neural Networks: A Comprehensive Foundation* (2nd ed.). Upper Saddle River, NJ: Prentice Hall. ISBN 0-13-908385-5.
- S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks", *IEEE Transactions on Neural Networks*, Vol 2, No 2 (Mar) 1991.

Categories: [Artificial neural networks](#) | [Computational statistics](#) | [Classification algorithms](#)
 | [Machine learning algorithms](#) | [Regression analysis](#)

This page was last modified on 31 August 2015, at 15:20.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

