Radial Basis Function Networks: Introduction

Neural Computation : Lecture 13

1. Introduction to Radial Basis Functions
2. Exact Interpolation
3. Common Radial Basis Functions
4. Radial Basis Function (RBF) Networks
5. Problems with Exact Interpolation Networks
6. Improving RBF Networks
7. The Improved RBF Network

Introduction to Radial Basis Functions

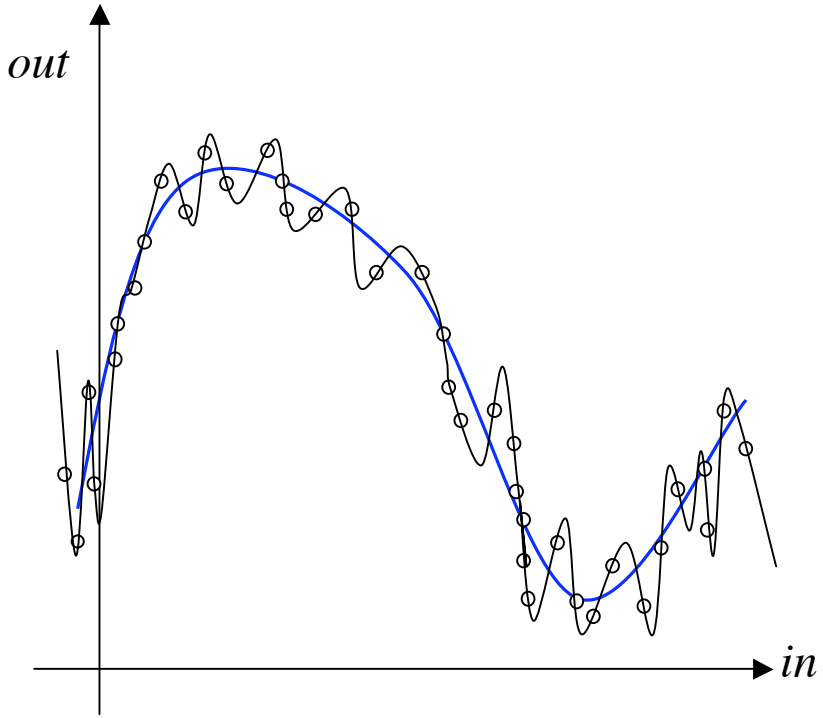
The idea of ***Radial Basis Function (RBF) Networks*** derives from the theory of function approximation. We have already seen how Multi-Layer Perceptron (MLP) networks with a hidden layer of sigmoidal units can learn to approximate functions. RBF Networks take a slightly different approach. Their main features are:

1. They are two-layer feed-forward networks.
2. The hidden nodes implement a set of radial basis functions (e.g. Gaussian functions).
3. The output nodes implement linear summation functions as in an MLP.
4. The network training is divided into two stages: first the "weights" from the input to hidden layer are determined, and then the weights from the hidden to output layer.
5. The training/learning is very fast.
6. The networks are very good at interpolation.

Now we'll spend the next three lectures studying the details…

Regression, Curve Fitting, Interpolation

We have already studied the general problem of identifying an underlying function from a set of noisy training data, and seen how Multi-Layer Perceptron networks can do that.



This is called ***regression*** or ***curve fitting***. The special case where the output function goes exactly through all the data points is called ***exact interpolation***.

Exact Interpolation

Formally, the ***exact interpolation*** of a set of N data points in a multi-dimensional space requires all the D dimensional input vectors $\mathbf{x}^p = \{x_i^p : i = 1,..., D\}$ to be mapped onto the corresponding target outputs t^p. The goal is to find a function $f(\mathbf{x})$ such that

$$f(\mathbf{x}^p) = t^p \qquad \forall\, p = 1,..., N$$

The radial basis function approach introduces a set of N ***basis functions***, one for each data point q, which take the form $\phi\left(\left\|\mathbf{x} - \mathbf{x}^q\right\|\right)$ where $\phi(\cdot)$ is some non-linear function whose form will be discussed shortly. Thus the qth such function depends on the distance $\left\|\mathbf{x} - \mathbf{x}^q\right\|$, usually taken to be Euclidean, between $\mathbf{x}$ and $\mathbf{x}^q$. The output of the mapping is then taken to be a linear combination of the basis functions, i.e.

$$f(\mathbf{x}) = \sum_{q=1}^{N} w_q \phi\left(\left\|\mathbf{x} - \mathbf{x}^q\right\|\right)$$

The idea is to find the "weights" w_q such that the function goes through the data points.

Equations Specifying the Weights

It is easy to determine equations for the weights by combining the above equations:

$$f(\mathbf{x}^p) = \sum_{q=1}^{N} w_q \phi\left(\left\|\mathbf{x}^p - \mathbf{x}^q\right\|\right) = t^p$$

The distances $\left\|\mathbf{x}^p - \mathbf{x}^q\right\|$ between data points p and q are fixed by the training data, so

$$\boldsymbol{\Phi}_{pq} = \phi\left(\left\|\mathbf{x}^p - \mathbf{x}^q\right\|\right)$$

is simply an array, or matrix, of training data dependent constant numbers, and the weights w_q are the solutions of the linear equations

$$\sum_{q=1}^{N} \boldsymbol{\Phi}_{pq} w_q = t^p$$

This can be written in matrix form by defining the vectors $\mathbf{t} = \{t^p\}$ and $\mathbf{w} = \{w_q\}$, and the square matrix $\boldsymbol{\Phi} = \left\{\boldsymbol{\Phi}_{pq}\right\}$, so the equation for $\mathbf{w}$ simplifies to $\boldsymbol{\Phi}\,\mathbf{w} = \mathbf{t}$.

Determining the Weights

It then follows that, provided the inverse of the matrix $\mathbf{\Phi}$ exists, any standard matrix inversion technique can be used to give the required weights:

$$\mathbf{w} = \mathbf{\Phi}^{-1}\,\mathbf{t}$$

in which the inverse matrix $\mathbf{\Phi}^{-1}$ is defined by $\mathbf{\Phi}^{-1}\mathbf{\Phi} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. There is no need to know exactly how to compute such inverse matrices, just that efficient computational algorithms and code exist that allow it to be done easily!

It can be shown that, for a large class of basis functions $\phi(\cdot)$, the matrix $\mathbf{\Phi}$ is indeed non-singular, and therefore invertible, as long as the data points are distinct.

Once the weights are determined, we have a function $f(\mathbf{x})$ that represents a continuous differentiable surface that passes exactly through each data point. The *Bias + Variance Decomposition* tells us that we need to be cleverer than that, but it is a good start…

Commonly Used Radial Basis Functions

A range of theoretical and empirical studies have indicated that many properties of the interpolating function are relatively insensitive to the precise form of the basis functions $\phi(r)$. Some of the most commonly used basis functions are:

1. Gaussian Functions:

$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \qquad \text{width parameter } \sigma > 0$$

2. Multi-Quadric Functions:

$$\phi(r) = \left(r^2 + \sigma^2\right)^{1/2} \qquad \text{parameter } \sigma > 0$$

3. Generalized Multi-Quadric Functions:

$$\phi(r) = \left(r^2 + \sigma^2\right)^{\beta} \qquad \text{parameters } \sigma > 0,\ 1 > \beta > 0$$

4. Inverse Multi-Quadric Functions:

$$\phi(r) = \left(r^2 + \sigma^2\right)^{-1/2} \qquad \text{parameter } \sigma > 0$$

5. Generalized Inverse Multi-Quadric Functions:

$$\phi(r) = \left(r^2 + \sigma^2\right)^{-\alpha} \qquad \text{parameters } \sigma > 0, \alpha > 0$$

6. Thin Plate Spline Function:

$$\phi(r) = r^2 \ln(r)$$

7. Cubic Function:

$$\phi(r) = r^3$$

8. Linear Function:

$$\phi(r) = r$$

Properties of the Radial Basis Functions

The Gaussian and Inverse Multi-Quadric Functions are ***localised*** in the sense that

$$\phi(r) \rightarrow 0 \quad \text{as} \quad |\mathrm{r}| \rightarrow \infty$$

but this is not strictly necessary. All the other functions above have the property

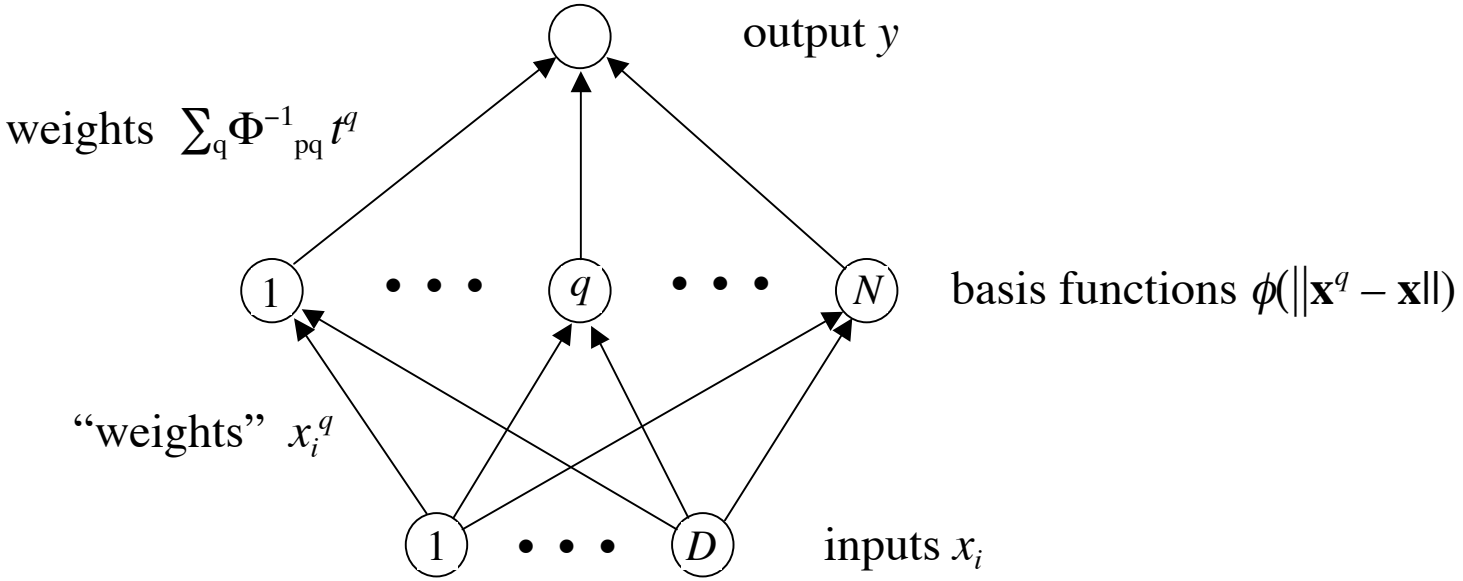
$$\phi(r) \rightarrow \infty \quad \text{as} \quad |\mathrm{r}| \rightarrow \infty$$

Note that in two or more dimensions, even the Linear Function $\phi(r) = r = \left\| \mathbf{x} - \mathbf{x}^p \right\|$ is non-linear in the components of $\mathbf{x}$. In one dimension, it leads to the piecewise-linear interpolating function which performs the simplest form of exact interpolation.

For neural network mappings, there are good reasons for preferring localised basis functions. We shall focus our attention on ***Gaussian basis functions*** since, as well as being localised, they have a number of other useful analytic properties. We can also see intuitively how to set their widths σ and build up function approximations with them.

Radial Basis Function Networks

You might think that what has just been described is not really a neural network. And a lot of people would agree with you! However, we can see how to make it look like one:



Note that the N training patterns $\{x_i^p, t^p\}$ determine the weights directly. The hidden layer to output weights multiply the hidden unit activations in the conventional manner, but the input to hidden layer weights are used in a very different fashion.

Example : Gaussian Radial Basis Functions

Suppose the chosen RBFs are Gaussians centred at the training data points $\{\mathbf{x}^p\}$. The function approximation $f(\mathbf{x})$ will then be built up as the weighted sum of Gaussians:

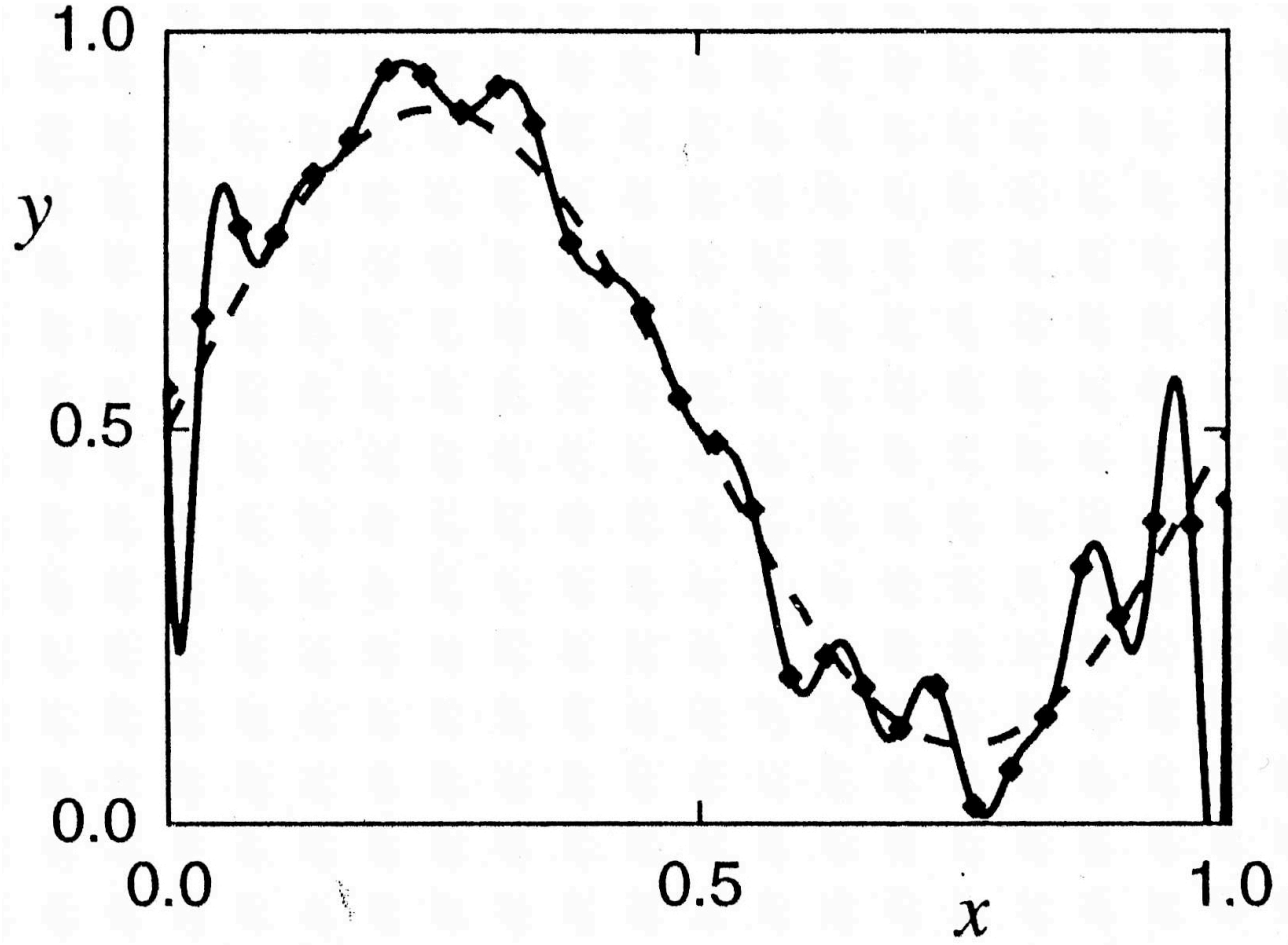
$$f(\mathbf{x}) = \sum_{p=1}^{N} w_p \phi_p(\mathbf{x}) = \sum_{p=1}^{N} w_p \exp\left(-\frac{\left\|\mathbf{x} - \mathbf{x}^p\right\|^2}{2\sigma^2}\right)$$

in which σ is the width of the Gaussians. If σ is too small compared to the distance between the data points, the approximation will consist of a narrow peak at each data point, which is not what we want. Having σ too large will be equally problematic. A convenient compromise is given by the average data-point separation d_{ave}

$$\sigma = 2d_{\text{ave}}$$

which ensures that on average the individual RBFs are neither too wide, nor too narrow, for the given training data. However, even then the noise in the training data can still lead to interpolation with massive errors in the function approximation.

Exact Interpolation with Gaussians of Width $\sigma = 2d_{ave}$



From: Neural Networks for Pattern Recognition, C. M. Bishop, Oxford University Press, 1995.

Problems with Exact Interpolation

We have seen how it is possible to set up RBF networks to perform exact interpolation, but there are two serious problems with these exact interpolation networks:

1. They perform poorly with noisy data

As has already been seen for Multi-Layer Perceptrons (MLPs), we do not usually want the network's outputs to pass through all the data points when the data is noisy, because that will be a highly oscillatory function that will not provide good generalization.

2. They are not computationally efficient

The network requires one hidden unit (i.e., one basis function) for each training data pattern, and so for large data sets the network will become very costly to evaluate. In particular, the matrix inversion cost is typically $O(N^3)$.

With MLPs, we can improve generalization by using more training data. The opposite happens in these RBF networks, and they take longer to compute as well.

Improving RBF Networks

To get better results, one can start with the basic structure of the RBF networks that perform exact interpolation, and improve upon them in a number of ways:

1. The number M of basis functions (hidden units) need not equal the number N of training data points. In general, it is better to have M much less than N.
2. The centres of the basis functions do not need to be defined as the training data input vectors. They can instead be determined by a training algorithm.
3. The basis functions need not all have the same width parameter σ. These can also be determined by a training algorithm.
4. Bias parameters can be introduced into the linear sum of activations at the output layer, as in an MLP. These will compensate for the difference between the average value over the data set of the basis function activations and the corresponding average value of the targets.

Of course, these will make analysing and optimising the network much more difficult.

The Improved RBF Network

When these changes are made to the exact interpolation formula, and the possibility of more than one output unit is allowed, one arrives at the RBF network mapping

$$y_k(\mathbf{x}) = w_{k0} + \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x})$$

which can be simplified by introducing an extra basis function $\phi_0 = 1$ to give

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x})$$

For the case of Gaussian basis functions, the hidden unit activations are

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

which involve $M \times D$ basis centre parameters $\{\boldsymbol{\mu}_j\}$ and M widths $\{\sigma_j\}$. Next lecture will look at how to determine all the network parameters $\{M, w_{kj}, \boldsymbol{\mu}_j, \sigma_j\}$.

Overview and Reading

1. We began by outlining the basic properties of RBF networks.
2. We then looked at the idea of exact interpolation using RBFs, and went through a number of common RBFs and their important properties.
3. Then we considered how to set up an RBF network to perform exact interpolation and noted two serious problems with it.
4. We ended by formulating a more useful form of RBF network.

Reading

1. Bishop: Sections 5.1, 5.2, 5.3, 5.4
2. Haykin-2009: Sections 5.1, 5.2, 5.3, 5.4
3. Gurney: Section 10.4
4. Callan: Section 2.6
5. Hertz, Krogh & Palmer: Section 9.7