Advanced Data Analytics
Lecture week 1: Neuroscale

Ian T. Nabney

University of BRISTOL

Overview

- Basic understanding of radial basis function networks
- Neuroscale as a non-linear topographic dimensionality reduction method
- Efficient training with shadow targets

University of BRISTOL

Radial Basis Function Networks

- The radial basis function (RBF) network is the main practical alternative to the multi-layer perceptron for non-linear modelling.
- Instead of units that compute a non-linear function of the scalar product of the input vector and a weight vector, the activation of the hidden units in an RBF network is given by a non-linear function of the distance between the input vector and a weight vector.
- One attraction of RBF networks is that there is a two-stage training procedure which is considerably faster than the methods used to train MLPs.
  1. The parameters of the basis functions are set so that they model the unconditional data density.
  2. Finding the optimal weights in the output layer is a quadratic optimisation problem, which can be solved efficiently using methods from linear algebra.
  3. The speed of training RBF networks also makes them attractive for use as a component in more complex models. The Generative Topographic Mapping (GTM) contains a non-linear map that must be retrained at each iteration of the training algorithm.

University of BRISTOL

RBF Definition

- The RBF mapping is given by

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

  where the ϕ_j are the basis functions, and the w_{kj} are the output layer weights.

- It is often convenient to absorb the bias weights into the summation by including an extra basis function ϕ_0 whose activation is the constant value 1

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x}). \tag{2}$$

- For a large class of basis functions, RBF networks are universal approximators.
- Once the ϕ_j are fixed, the output is a linear function of the weights.
- The NETLAB implementation of radial basis function networks includes a choice of three different basis functions, written as a function of the radial distance $r = \|x - \mu_j\|$.

University of BRISTOL

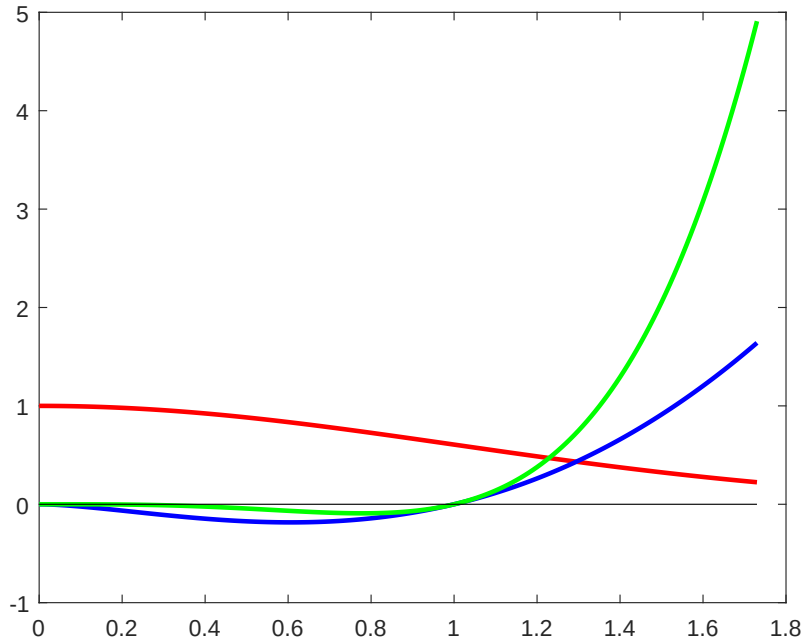
Choice of basis functions: Netlab

- Gaussian This is left unnormalised, $\exp(-r^2/\sigma^2)$, as the output weights can provide any scaling required.
- Thin plate spline The function $r^2 \log(r)$ is an unbounded function that takes on negative values for $0 < r < 1$. It is easy to show that

$$\lim_{r \to 0} r^2 \log(r) = 0. \qquad (3)$$

- $r^4 \log r$ This is also an unbounded function that takes on negative values for $0 < r < 1$. It is easy to show that

$$\lim_{r \to 0} r^4 \log(r) = 0. \qquad (4)$$

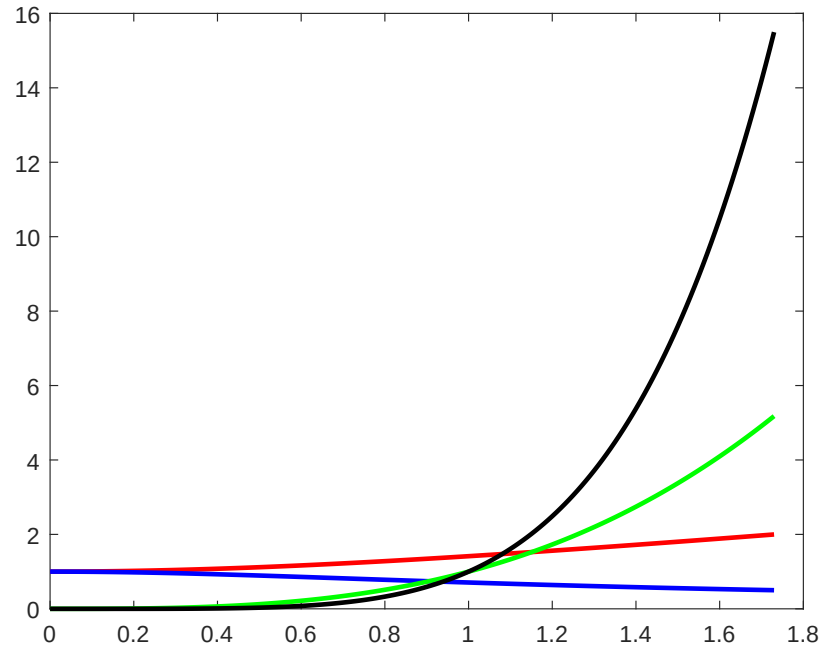


RBF basis functions. Squared exponential (red), thin plate spline (blue) and $r^4 \log r$ (green).

Choice of basis functions: Scipy

In addition to linear, Gaussian and thin plate spline

- Multi-quadric $\sqrt{r^2+1}$
- Inverse multi-quadric $\frac{1}{\sqrt{r^2+1}}$
- Cubic r^3
- Quintic r^5



RBF basis functions in Scipy. Multi-quadric (red), inverse multi-quadric (blue), cubic (green) and quintic (black).

Training an RBF

- It is possible to choose good (though possibly not optimal) parameters for the hidden units without having to perform a full non-linear optimisation of all the network parameters.
- The sum of the basis functions $\sum_{j=0}^{M} \phi_j$ represents the unconditional density of the input data.
- Therefore we can use an unsupervised learning procedure for choosing the basis function parameters, and a supervised method for optimising the output layer weights.

Basis function selection

- To ensure that the basis functions as a whole approximate probabilities, we can use positive definite normalised basis functions which are themselves density functions: in this case, we have a mixture model formalisation.
- One obvious choice for the basis functions is therefore a Gaussian.
- However, there is a drawback to this choice: the resulting density estimator is necessarily biased for a finite set of samples. If we allow non-negative basis functions, then the asymptotic convergence of the bias (as the number of samples n tends to infinity) is faster than if we insist on positive definite basis functions.

Basis function training

- Aim is to choose the basis function centres $\mathbf{c}_j$ and, where appropriate, widths $\mathbf{wi}_j$ (for Gaussian basis functions).
- The simplest procedure, which is surprisingly successful in practice, is simply to choose a subset of the data points at random and use these as the basis function centres $\mathbf{c}_j$.
- A more sophisticated approach is to cluster the data into an appropriate number of clusters, and then to use the cluster centres for the $\mathbf{c}_j$.
- As the aim is to model the data density with a linear combination of basis functions, an obvious approach is to treat the basis functions as a mixture model and to use the EM algorithm to find the parameters.
- Practical experience has shown that setting the widths of the basis functions equal to the variances of the corresponding mixture model tends to give poor generalisation results because the widths are too small and there is insufficient overlap between the basis functions.

Training output layer weights

- We can write (2) in matrix form as

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{\phi}\mathbf{W}, \tag{5}$$

where $\mathbf{W} = (w_{kj})$ is the output layer weight matrix and $\boldsymbol{\phi} = (\phi(j))$ is the matrix of hidden unit activations.

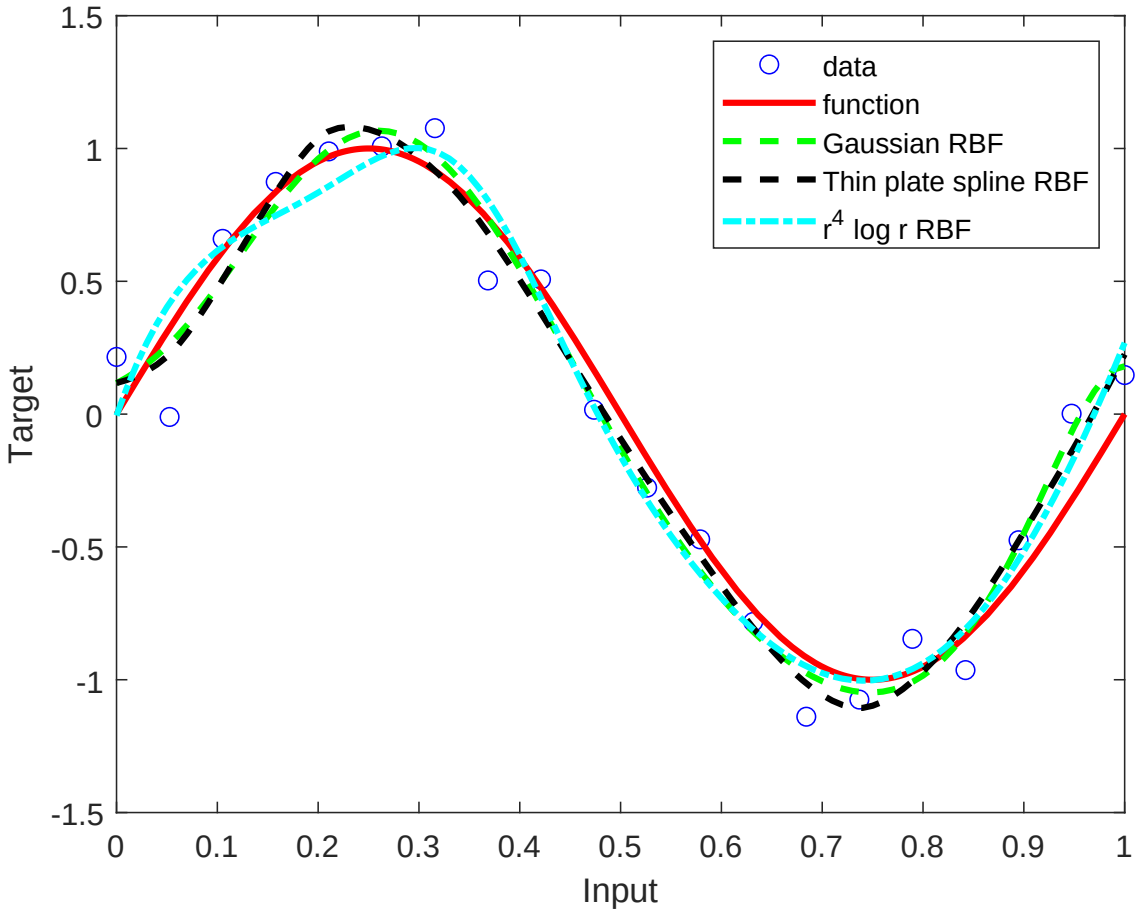
- Suppose that we have a data set $\mathbf{X}$: then we can extend (5) and write

$$\mathbf{Y}(\mathbf{X}) = \boldsymbol{\Phi}\mathbf{W}, \tag{6}$$

where $\boldsymbol{\Phi}$ is the design matrix. We use a sum-of-squares error function: since this error function is quadratic in the weights, its minimum can be found using the pseudo-inverse of the design matrix:

$$\mathbf{W} = \boldsymbol{\Phi}^{\dagger}\mathbf{T}. \tag{7}$$

Regression case study



Justification for Neuroscale

The Sammon mapping has a number of disadvantages:

1. The number of parameters in the optimisation problem grows linearly with the size of the dataset.
2. The mapping is defined only for the original data points $\mathbf{x}_i$.
3. The only way to compute the mapping for a new point $\mathbf{x}^*$ is to add it to the dataset and reoptimise the stress measure (though most of the $\mathbf{y}_i$ can be started at their earlier values to cut down on some of the computational cost).
4. As the mapping does not generalise, it is impossible to take a sub-sample of the training data to train the map more efficiently.

To get round these difficulties, the Neuroscale model was introduced by Tipping and Lowe. This model defines a nonlinear map $\mathbb{R}^d \to \mathbb{R}^q$ using an RBF, since a particularly efficient training algorithm is available.

Training Neuroscale

- We have seen how the output layer weights of an RBF can be trained very efficiently with a single pass matrix pseudo-inverse computation. This advantage is lost here, since the error function contains quartic terms.
- However, there is a training algorithm that is more efficient than using a general non-linear optimiser.
- This algorithm, called shadow targets, makes use of the special form of the error function and the linear dependence of the network output on the output layer weights.
- It is based on a model trust region approach, analogous to that used in the scaled conjugate gradient algorithm.

Shadow Targets

We can express the necessary derivatives (for output dimension r) in the form

$$\boldsymbol{\nabla} E = \boldsymbol{\Phi}^T \mathbf{e}_r, \tag{8}$$

where

$$\boldsymbol{\nabla} E = \left(\frac{\partial E}{\partial w_{1r}}, \frac{\partial E}{\partial w_{2r}}, \ldots, \frac{\partial E}{\partial w_{Mr}} \right)^T \quad \text{and} \quad \mathbf{e}_r = \left(\frac{\partial E}{\partial \mathbf{y}_{1r}}, \frac{\partial E}{\partial \mathbf{y}_{2r}}, \ldots, \frac{\partial E}{\partial \mathbf{y}_{Mr}} \right)^T.$$

For normal RBF regression, the same equation is valid but with

$$\frac{\partial E}{\partial \mathbf{y}_i} = (\mathbf{y}_i - \mathbf{t}_i). \tag{9}$$

The key idea of the shadow targets algorithm is to use (9) to estimate hypothetical targets $\hat{\mathbf{t}}_i$.

$$\begin{aligned} \hat{\mathbf{t}}_i &= \mathbf{y}_i - \frac{\partial E}{\partial \mathbf{y}_i} \\ &= \mathbf{y}_i + 2 \sum_{j \neq i} \left(\frac{d_{ij} - d_{ij}^*}{d_{ij}} \right) (\mathbf{y}_i - \mathbf{y}_j). \end{aligned} \tag{10}$$

University of BRISTOL

Estimating shadow targets

- For a fixed set of targets, the least squares problem can be solved directly:

$$\mathbf{W} = \mathbf{\Phi}^{\dagger}\hat{\mathbf{T}}. \tag{11}$$

  Of course, for our problem the estimated targets $\hat{\mathbf{t}}_i$ are not fixed, since $\partial E/\partial \mathbf{y}_i$ depends on the network weights.

- We iterate this procedure, re-estimating the shadow targets at each step. However, in the early stages of training, the targets estimated by (10) may be poor, and hence the weights given by (11) may increase the error.

- It is therefore more practical to trust the approximation given by (10) to a limited extent and to increase our trust only when E_{sam} decreases. This is achieved by introducing an additional parameter η and estimating the targets by

$$\hat{\mathbf{t}}_i = \mathbf{y}_i - \eta \frac{\partial E}{\partial \mathbf{y}_i}. \tag{12}$$

  It is clear that η should be restricted to the range $(0, 1)$.

University of BRISTOL

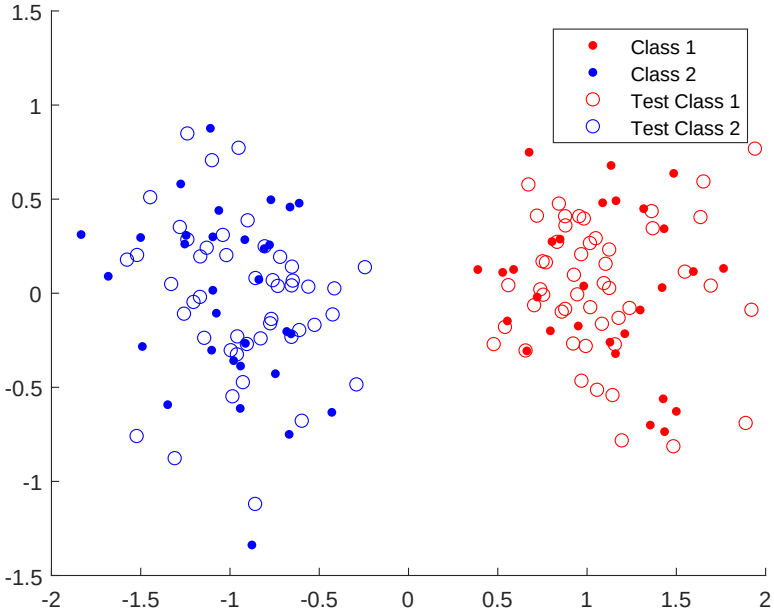
Full algorithm

1. Initialise the weights $\mathbf{W}$ to small random values.
2. Initialise η to some small positive value.
3. Calculate $\mathbf{\Phi}^{\dagger}$. This is computationally costly, but needs to be done only once.
4. Use (12) to compute estimated targets $\hat{\mathbf{t}}_i$.
5. Solve for the weights $\mathbf{W} = \mathbf{\Phi}^{\dagger}\hat{\mathbf{T}}$.
6. Calculate E_{sam} and compare with previous value.
    1. If E_{sam} has increased, set $\eta = \eta \times k_{\text{down}}$. Restore previous values of $\mathbf{W}$.
    2. If E_{sam} has decreased, set $\eta = \eta \times k_{\text{up}}$.
7. If convergence has not been achieved, return to Step 4.

University of BRISTOL

Neuroscale demonstration

- Generate 60 data points from a mixture of two Gaussians in 4 dimensional space.
- Train an RBF with 10 centres using thin plate spline functions using shadow targets (converges in 7 iterations).
- Plot points in 2D with red (class 1) and blue (class 2) dots.
- Generate a further 100 points from the original distribution and project them.



University of BRISTOL