An Overview of Radial Basis Function Networks

Joydeep Ghosh and Arindam Nag

Department of Electrical and Computer Engineering,
The University of Texas, Austin, TX 78712

Abstract: This chapter presents a broad overview of Radial Basis Function Networks (RBFNs), and facilitates an understanding of their properties by using concepts from approximation theory, catastrophy theory and statistical pattern recognition. While this chapter is aimed to provide an adequate theoretical background for the subsequent application oriented chapters in this book, it also covers several aspects with immediate practical implications: alternative ways of training RBFNs, how to obtain an appropriate network size for a given problem, and the impact of the resolution (width) of the radial basis functions on the solution obtained. Some prominent applications of RBFNs are also outlined.

1 Introduction

A Radial Basis Function (RBF), ϕ, is one whose output is symmetric around an associated *center* , $\boldsymbol{\mu}_c$. That is, $\phi_c(\mathbf{x}) = \phi(||\mathbf{x} - \boldsymbol{\mu}_c||)$, where $||\cdot||$ is a vector norm. For example, selecting the Euclidean norm and letting $\phi(r) = e^{-r^2/\sigma^2}$, one sees that the Gaussian function is an RBF. Note that Gaussian functions are also characterized by a *width* or scale parameter, σ, and this is true for many other popular RBF classes as well. So such classes can be represented as:

$$\phi_c(\mathbf{x}) = \phi(||\mathbf{x} - \boldsymbol{\mu}_c||; \sigma). \tag{1}$$

A set of RBFs can serve as a basis for representing a wide class of functions that are expressible as linear combinations of the chosen RBFs:

$$y(\mathbf{x}) = \sum_{j=1}^{M} w_j \phi(||\mathbf{x} - \boldsymbol{\mu}_j||). \tag{2}$$

A Radial Basis Function Network (RBFN) (Fig. 1), is nothing but an embodiment of Eq. 2 as a feedforward network with three layers: the inputs, the hidden/kernel layer and the output node(s). Each hidden unit represents a single radial basis function, with associated center position and width. Such hidden units are sometimes referred to as centroids or kernels. Each output unit performs a weighted summation of the hidden units, using the w_js as weights.

The issue of finding the individual components given a superposition of Gaussians was explored in the early sixtees [Meg61]. Subsequently, the use of RBFs for numerical interpolation and function approximation was demonstrated [Pow85, BL88]. Neural network formulations of this approach led to renewed interest in these functional forms [LK88, MD89, PG90].

Part of the excitement was due to analogies with *localized receptive fields* found in several biological structures including the visual cortex and the ganglion cells. This further motivated the study of bell-shaped functions which take on high values only in the neighborhood of a specific input. Powerful *universal* approximation properties of Eq. 2 were demonstrated for various settings [KHK90, PS91, PS93], and convergence rates were established [NG96, XKY94]. By decoupling the estimation of the center locations and widths ($\boldsymbol{\mu}_j$s and σ_js) from the weights (w_js) to the output layer, fast and practical training algorithms were developed to realize suitable networks given a set of possibly noisy training data. At the same time, useful ties were shown between RBFNs and regularization theory, kernel regression, Bayesian classification etc. At present, RBFNs are perhaps second in popularity only to multilayered perceptrons as the neural network of choice for realizing static maps for a variety of function approximation and classification problems. Many of the advantages of RBFNs will be evident in the following sections.

2 Exact Interpolation

Consider the exact interpolation problem: given a set of N different d-dimensional input vectors $\mathbf{x}^n$ and a corresponding set of one-dimensional targets t^n, find a continuous function $h(\mathbf{x})$ such that

$$h(\mathbf{x}^n) = t^n, \quad n = 1, \ldots, N. \tag{3}$$

The solution to this problem by adopting the radial basis function approach consists of choosing a set of N *basis functions*, centered at the N data points, and using the functional form of Eq. 1 giving :

$$h(\mathbf{x}) = \sum_{n=1}^{N} w_i \phi(||\mathbf{x} - \mathbf{x}^n||). \tag{4}$$

An exact solution is possible if one can solve the following set of N simultaneous linear equations to find the unknown coefficients(weights) of the expansion :

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} \tag{5}$$

where

$$\phi_{ij} = \phi(||\mathbf{x}^i - \mathbf{x}^j||), \quad i, j = 1, 2, \ldots, N. \tag{6}$$

The compact matrix form of Eq. 5 is

$$\mathbf{\Phi w} = \mathbf{t}. \tag{7}$$

It has been shown by Michelli [Mic86] and more recently by Light [Lig92] that there exist a large class of functions $\phi(\cdot)$, including Gaussians, inverse multiquadrics and thin-plate splines, for which the interpolation matrix $\mathbf{\Phi}$ is non-singular provided the data points are distinct. Hence we may solve for Eq. 7 to give

$$\mathbf{w} = \mathbf{\Phi^{-1}t}, \tag{8}$$

and achieve exact interpolation. This is because, when the weights obtained by Eq. 8 are substituted in Eq. 4, the function $h(\mathbf{x})$ represents a continuous differentiable surface passing through each data point.

Both theoretical studies and empirical results show that many characteristics of the interpolating function are relatively insensitive to the precise type of the nonlinearity, ϕ. Relatively non-intuitive functions such as thin plate splines, which have a global response, have been used successfully for interpolation. In fact, the form of ϕ can itself be adapted during training, though this typically does not yield extra power unless the number of kernels is small [WS96]. Gaussian RBFs are the most popular choices for ϕ [LK88]. Their localized response helps in the decoupling of the learning process as will be discussed later and is also more biologically plausible. Gaussian RBFs have a number of other useful analytical properties, as will be apparent later on.

Generalization to multidimensional target space is straightforward. The mapping from d-dimensional input space $\mathbf{x}$ to k-dimensional target space $\mathbf{t}$ is given by

$$h_k(\mathbf{x}^n) = t_k^n, \quad n = 1, \ldots, N, \tag{9}$$

where t_k^n are components of the output vector $\mathbf{t}^n$ and $h_k(\mathbf{x})$ are obtained by linear superposition of the same N basis functions as used for the one-dimensional output case:

$$h_k(\mathbf{x}) = \sum_n w_{kn} \phi(||\mathbf{x} - \mathbf{x}^n||). \tag{10}$$

The weight parameters are obtained in the form

$$w_{kn} = \sum_j (\mathbf{\Phi}^{-1})_{nj} t_k^j \tag{11}$$

where the same matrix $\mathbf{\Phi}^{-1}$ is used for each of the output functions.

3 Function Approximation

In practice, one is typically not interested in exact interpolation. Firstly, data is usually noisy, and an interpolating function passing through every data point will lead to overfitting and thereby poor generalization. Better generalization will be achieved using a smoother fit that averages out the noise in the data. Secondly, to guarantee exact interpolation, the number of basis functions required is equal to the number of patterns in the learning data set. Thus it is apparent that for a large data set the mapping function can become very costly to evaluate.

A radial basis function model for function approximation and generalization is obtained by modifying the exact interpolation procedure(Broomhead and Lowe, 1988; Moody and Darken, 1989). This model gives a smoother fit to the data using a reduced number of basis functions which depends on the complexity of the mapping function rather than the size of the data set. The modifications are as follows:

1. Number of basis functions : The number of basis functions M is typically less than N.
2. Centers of basis functions : The centers of the basis functions are determined as a part of the training process, rather than being constrained to be located at each input data point.
3. Width of the basis functions : The training process may adapt the width parameter σ_j for each of the basis functions, instead of assigning the same width parameter σ to all the basis functions.
4. Addition of bias parameters : Bias in the form of a constant may be added to Eq. 2 to compensate for the difference between the average value over the data set of the basis function activations and the corresponding values of the targets.

By applying these modifications to the exact interpolation formula, the form of the radial basis function network mapping obtained is

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}. \tag{12}$$

The biases w_{k0} can be incorporated into the summation by introducing an extra basis function ϕ_0 and setting its activation to unity, so that one can write:

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x}) \tag{13}$$

or, in matrix form,

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{W}\boldsymbol{\phi}. \tag{14}$$

The equation of a Gaussian basis function is

$$\phi_j(\mathbf{x}) = e^{\left(-\frac{||\mathbf{x}-\boldsymbol{\mu}_j||^2}{2\sigma_j^2}\right)} \tag{15}$$

where $\mathbf{x}$ is the d-dimensional input vector with elements x_i and $\boldsymbol{\mu}_j$ is the center of the basis function ϕ_j and has elements μ_{ji}.

The property of *universal approximation* by linear superposition of Gaussian basis functions, in which the widths of the Gaussians are treated as adjustible parameters was proved by Hartman et al.(1990). More general results were obtained by Park and Sandberg(1991,1993) who show that, with only mild restrictions on the form of the kernel functions, the universal approximation property still holds. These proofs are existence proofs relying on the availability of arbitrarily large number of hidden units, and do not offer practical procedures for the construction of networks. Nevertheless, these theorems are crucial in providing a theoretical foundation on which practical applications can be based with confidence. Any parametric model used for function approximation, defines a space of approximating functions. A model having a unique set of parameters which best approximates the underlying function is said to possess the property of *best approximation*. Radial basis function networks exhibit this property [PG90], which is not shown by multilayered perceptrons.

For arbitrary covariance matrices $\mathbf{\Sigma}_j$, Gaussian radial basis functions, or more appropriately, "elliptical" basis functions, have the form

$$\phi_j(\mathbf{x}) = e^{\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right\}} \tag{16}$$

The number of independent adjustible parameters for each basis function is $d(d+3)/2$ for symmetic covariance matrices $\mathbf{\Sigma}_j$, where d is the dimensionality of the input space. This is in comparison with the $(d+1)$ independent parameters for the basis functions. If σ_j is also adapted during training, the resultant network is less susceptible to noise, as less important dimensions can be tuned out [BG92]. However, more fine-tuning of the learning rates needs to be done for good results. Thus one of the design issues is the consideration of the tradeoff involved between the use of smaller number of basis functions with many adjustible parameters versus a larger number of less flexible functions.

3.1 Convergence Rates

Given that RBFNs can uniformly approximate any continuous function to an arbitrary precision if enough hidden units are provided, a natural question arises regarding the convergence rate, i.e., how does the accuracy of approximation improve with the number of hidden units, M. Clearly, the answer depends on the class of functions being considered for approximation, as well as the type of basis functions used. One can show that the kernel regression estimator - an extension of the Parzen window estimator for statistical regression problems - is expressible as a *normalized* RBFN (see Sec. 7) using Gaussian basis functions. By using the theoretical results already known about kernel regression estimators, one readily obtains a number of interesting convergence results for (normalized) RBFNs [XKY94]. For example, an upper bound on the L_2 convergence rate for approximating a function in the function class that satisfies a Lipshitz condition of order α, $0 < \alpha \leq 1$ is $O(M^{-2\alpha/(2\alpha+d)})$, where d is the input dimension. Similarly, for the class of functions that have order q ($q \geq 1$) derivatives that are square integrable, the corresponding bound is $O(M^{-2q/(2q+d)})$. As expected, the more smooth the functions are (the larger the α or q), the faster is the rate. In particular, if the ratio d/q is held constant, the convergence rate does not depend on the input dimensionality. Thus, for this class of functions, the curse of dimensionality can be avoided. Furthermore, if $q >> d$, the rate appoaches O(M).

In the results stated above, the number of training samples does not figure, since estimation error is not accounted for. In other words, the results are for the best possible choice of the parameters. In practice, given a finite training set and a heuristic learning rule, the parameters obtained may not be optimal. Recognizing that the total generalization error is the sum of estimation and approximation errors, Niyogi and Girosi [NG96] provide bounds that incorporate the number of training samples available.

A powerful approach to balancing estimation and approximation errors is complexity regularization. When applied to a nested family of functions of adequate power (including RBFs with increasing number of kernel units), complexity regularization aims to automatically choose a suitable network size depending on the number of training samples and the VC dimension of the problem, and then directly minimize the empirical risk. Using this framework and several reasonable constraints, strong consistency results [KLL96] and $O(\sqrt{\frac{\log n}{n}})$ convergence rates have been proved for RBF networks [KL97].

4 Radial Basis Function Network Training

From Eq. 2, one can see that designing an RBFN involves selecting the type of basis functions, ϕ with associated widths, σ, the number of functions, M, the center locations $\boldsymbol{\mu}_j$ and the weights w_j. Typically Gaussians or other *bell-shaped* functions with compact support are used, though thin-plate splines have performed better for some function approximation problems. The choice of M is related to the important problem of model order selection, and is treated in detail in Sec. 5. Given that the number of basis functions and their type have been selected, training an RBFN involves determining the values of three sets of parameters: the centers, the widths and the weights, in order to minimize a suitable cost function. In general, this is a non-convex optimization problem.[1]

4.1 Supervised Training

One can perform stochastic gradient descent on the cost function to iteratively update the parameters, once per training sample presentation. As an illustration, consider the sum-of-squares cost function given by

$$E = \sum_n E_n \tag{17}$$

[1] In certain special cases, such as classification problems in which the patterns are separable by hyperspheres, a cost function that is local minima free with respect to all the weights, can be obtained [BFG95].

such that:

$$E_n = \frac{1}{2} \sum_k \{t_k^n - y_k(\mathbf{x}^n)\}^2, \tag{18}$$

where t_k^n is the target value of output unit k when the network is presented with input vector $\mathbf{x}^n$. If Gaussian basis functions are used to minimize this cost function, one readily obtains the update equations[GDB92]:

$$\Delta w_{kj} = \eta_1 (t_k^n - y_k(\mathbf{x}^n)) \phi_j(\mathbf{x}^n) \tag{19}$$

$$\Delta \boldsymbol{\mu}_j = \eta_2 \phi_j(\mathbf{x}^n) \frac{||\mathbf{x}^n - \boldsymbol{\mu}_j||}{\sigma_j^2} \sum_k \{y_k(t_k^n) - \mathbf{x}^n\} w_{kj} \tag{20}$$

$$\Delta \sigma_j = \eta_3 \phi_j(\mathbf{x}^n) \frac{||\mathbf{x}^n - \boldsymbol{\mu}_j||^2}{\sigma_j^3} \sum_k \{y_k(t_k^n) - \mathbf{x}^n\} w_{kj} \tag{21}$$

where η_1, η_2, η_3 are the learning rates.

4.2 Two-stage Training

The simultaneous update of all three sets of parameters using Eq. 19, 20, 21 may be suitable for non-stationary environments or on-line settings. But for static maps, RBFNs with localized basis functions offer a very attractive alternative, namely that in practice, the estimation of parameters can be decoupled into a *two-stage procedure*:
(i) determine the $\boldsymbol{\mu}_j$s and σ_js, and
(ii) for the centers and widths obtained in step (i), determine the weights to the output units.
Both sub-problems allow for very efficient batch mode solutions. Moreover, for many situations, this decoupling leads to little loss in the quality of the final solution as compared to the optimal solution. In fact, given finite training data and computational resources, they often provide better solutions than those obtained by attempting to simultaneously determine all three sets of parameters.

In the first stage only the input values $\{\mathbf{x}^n\}$ are used for determining the centers $\boldsymbol{\mu}_j$ and the widths σ_j of the basis functions. Thus learning is unsupervised and can even use unlabelled data. Once the basis function parameters are fixed, supervised training (i.e.training using target information)can be employed for determining the second layer weights. We shall now consider the two stages in more detail.

4.2.1 Unsupervised Training of basis function centers and widths

The key to determining the location and widths of the *localized* basis functions is to view them as representing the input data density. Since one would like at least one hidden unit to be active in regions of the input space where data is present, basis functions with localized responses should be placed in regions of data concentration, and the widths should be related to the local spread of data and the distance to nearby centroids. This viewpoint suggests the following alternatives:

(i) **Random subset selection**: The simplest way of choosing the radial basis function centers $\boldsymbol{\mu}_j$ is to locate the centers at a subset of the data points, chosen randomly from the training data set. This choice is sensitive to how representative are the selected data points of the overall population. However it provides an initial starting point from which various adaptive techniques can be used to tune the center locations and, optionally, to change the number of centers used.

(ii)**Clustering Algorithms**: A better approach is to cluster the input vectors and then locate the basis functions at the cluster centers. A variety of clustering techniques can be used. Suppose we need to partition N data points $\mathbf{x}^n$ into K clusters and find the corresponding cluster centers. The K-means algorithm seeks to partition the data points into K subsets S_j by minimizing the sum of squares clustering function:

$$J = \sum_{j=1}^{K} \sum_{n \epsilon S_j} ||\mathbf{x}^n - \boldsymbol{\mu}_j||^2 \tag{22}$$

where $\boldsymbol{\mu}_j$ is the geometric mean of the data points in the subset S_j and is given by

$$\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{n \epsilon S_j} \mathbf{x}^n. \tag{23}$$

One method of finding these clusters is by using the batch version. First the data points are randomly assigned to K subsets. The centers for each of the subsets is then computed. The data points are then reassigned to the cluster whose center is nearest. The procedure is repeated till there are no further changes in the grouping. Alternatively one can use a stochastic online process, where initial centers are randomly chosen from the training data, and are then updated as each data is presented sequentially, using the rule

$$\Delta \boldsymbol{\mu}_j = \eta(\mathbf{x}^n - \boldsymbol{\mu}_j) \tag{24}$$

where η is the learning rate parameter. Other unsupervised techniques such as Kohonen's self organizing feature map can also be used for determining the basis function centers.

(iii)**Mixture models**: A probability density function $p(\mathbf{x})$ can be modelled as a superposition of component densities as

$$p(\mathbf{x}) = \sum_{j=1}^{M} P(j)\psi_j(\mathbf{x}) \tag{25}$$

where P is the prior probability of the component density ψ_j. The parameters of such a mixture model can be obtained using maximum likelihood [Bis95]. If the component densities are bell-shaped, they indicate local peaks in density and suggest placement of basis functions at their locations.

(iv)**Width Determination**: The basis function widths can either be chosen to be the same for all the units or can be chosen differently for each unit depending on the part of the input space they represent. In the first case, the common width can be set as some multiple of the average distance between the basis centers. This multiple governs the amount of smoothing; small widths lead to less smooth functions. In the second case each center's width is set to a multiple typically 1.5 to 2 times of the average distance to L nearest neighbors. While the widths can be adjusted during subsequent training using Eq. 21 or other techniques, many researchers do not do so and prefer the simplicity resulting from a smaller (though sub-optimal) search space.

The widths play an important part in the nature of the function realized by an RBFN, and leads to qualitatively different behaviors when varied over a large space. This issue is further considered in Sec. 6.

4.2.2 Batch training of output layer weights

Once the basis parameters are determined, the transformation between the inputs and the corresponding outputs of the hidden units is fixed. The network can thus be viewed as an equivalent single-layer network with linear output units. Minimization of the sum-squared error function (Eq. 17) yields the well-known least-squares solution for the weights:

$$\mathbf{W}^T = \mathbf{\Phi}^{\dagger}\mathbf{T} \tag{26}$$

where $(\mathbf{T})_{nk} = t_k^n$, $(\mathbf{\Phi})_{nj} = \phi_j(\mathbf{x}^n)$, and $\mathbf{\Phi}^{\dagger} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T$ denotes the pseudo-inverse of $\mathbf{\Phi}$. $\mathbf{\Phi}$ is referred to as the *design matrix* and $\mathbf{A} = \mathbf{\Phi}^T\mathbf{\Phi}$ is called the *variance matrix.* Thus the weights can be obtained by direct batch-moded matrix inversion techniques. In practice, to avoid the possible problems due to ill-conditioning of the matrix $\mathbf{\Phi}$, singular value decomposition is usually used to solve the equation.

4.3 Comparison of Two-Stage Training with Supervised Training

Neglecting the output data, while determining the basis function centers and widths using the above mentioned unsupervised training techniques leads to sub-optimal choices of these parameters. Supervised training may lead to optimal estimation of the centers and widths. However there are a number of disadvantages with the latter scheme. Firstly, the gradient descent method discussed earlier is a non-linear optimization technique and is computationally expensive. However if the basis functions are well localized then only few basis functions will generate significant activation. Efforts to determine these units and only using them in the subsequent computation can reduce the training time. Another disadvantage of this kind of training is that there is no guarantee that the basis functions will remain localized. Finally the main attraction of radial basis function network is its two stage learning, which is lost while using supervised training.

4.4 Variants

For classification problems, each local basis function is typically preferentially responsive to a certain class. Based on this obervation, an iterative hybrid learning algorithm was proposed in which the center locations are updated by LVQ while the output weights are updated in the usual way [GC94, BP96]. Another possibility is to select the kernel(s) to be updated in a probabilistic rather than deterministic manner[OF94]. In [BP96], ideas from robust statistics were leveraged to determine the center locations and widths using the marginal median and absolution deviations of the (windowed) data points assigned to each kernel. Finally, a subtle link between unsupervised and supervised learning has been shown in [GC94]. If the output weights are held constant and a fake (constant) output target is provided for all samples, the kernel locations adapt in a manner very similar to Kohonen's self-organizing feature map.

5 Model Selection

The primary goal of network training is not to find an exact fit to the training data but to model the statistical process responsible for generating the data. Thus the emphasis is on generalization or performance on data outside the training set. The key to valid generalization is to select a generalizer whose inductive bias is suited to the data, and then to match the model complexity with the complexity of the underlying map generating the data.

The problem of model selection can be understood as a trade-off between bias and variance [GBD92]. The generalization error can be decomposed into the sum of bias squared and variance. A typical trade-off between these two components of generalization error is a function of the model complexity [GBD92]. Too simple a model will have a high bias in the sense that the model, on the average will differ considerably from the desired one, even though specific instances of the model, obtained by changing the training data, initialization conditions etc, may hardly differ from one another. On the other hand, too complex a model may have a low bias but will have high variance and be prone to the specificities of the finite training set.

The balancing of bias and variance can be viewed as a problem of finding the right number of free parameters. For RBFNs, this involves trying to determine the optimal number of hidden units. Hence the analysis of the effect of adding a new hidden unit or removing an existing unit is an important one. A brute-force approach is to train several networks using different numbers of hidden units and/or different number of training samples. Then, one of the networks can be chosen based on cross-validation or some other criteria such as AIC or GCV [Moo94].

Fortunately, the localized nature of RBFs and the decoupled training methods suggest superior alternatives to brute-force model selection. Two main approaches to model selection have been taken for RBFNs: one may choose an adequate number of basis functions, but then add a regularization term to the cost function to enforce smoothness or some other desirable characteristic of the function to be realized. Similar effects can be achieved through early stopping of the training process. The second approach is to either add or delete hidden units as training proceeds, leading to growing or pruning algorithms [GT94, Ree93]. These two approaches are examined in the next two sections.

5.1 Regularization of RBFNs

Regularization is a powerful technique to favor certain solutions over others by adding a penalty functional to the original cost function. The penalty embodies *Occam's Razor* in some form by penalizing more complex solutions (ridge regression, subset selection) or solutions that are less smooth (non-parametric penalty functionals using a differential operator). A variety of penalties are studied in [Fri94]. Regularization can be linked to *weight decay* [KH92, CDS90] and to the use of model priors in the Bayesian framework [Mac95]. Complexity regularization followed by empirical risk minimization has been studied because of nice convergence properties, as mentioned in Sec. 3.1.

Below, we outline how the linear nature of the second-stage training allows one to apply well known solutions to ridge regression in linear systems, for determining the weights of an RBFN with regularization. For details and a very readable exposition, see [Orr95, Orr96].

5.1.1 Projection Matrix

Consider an RBFN with M hidden units, 1 output unit and $N > M$ training samples. For each of the hidden units, one can associate a N dimensional vector whose i^{th} component is the unit's response to the i^{th} training sample. Since the network output is a linear combination of the hidden unit outputs, the possible output vector lies in a M dimensional space spanned by the M N - dimensional hidden unit vectors. However the training set output vector $\mathbf{t}$ lives in a N dimensional space. The closest output vector to $\mathbf{t}$ (in L_2 norm) is nothing but the projection of $\mathbf{t}$ onto the M dimensional subspace. The sum squared error for this solution is

$$E = \mathbf{t}^T \mathbf{P}^2 \mathbf{t} \tag{27}$$

with the projection matrix

$$\mathbf{P} = \mathbf{I}_N - \mathbf{\Phi} \mathbf{A}^{-1} \mathbf{\Phi}^{\mathbf{T}} \tag{28}$$

where $\mathbf{I}_N$ is an identity matrix of dimension N, $\mathbf{\Phi}$ is the matrix of basis functions and $\mathbf{A} = \mathbf{\Phi}^T \mathbf{\Phi}$. The projection matrix will prove useful in methods such as ridge regression and forward selection described later.

5.1.2 Cross-Validation

The cross-validation technique divides the available data measurements into two sets, the training set and the test set. The training set is used for determining the network parameters and the test set is used to compute the prediction error. However this may introduce a bias since we are relying on one particular division of the data into training and test samples. A better way is to partition the original data in several different ways, each time training the network on training samples and computing the prediction error. We could then obtain an unbiased estimate of the prediction error by averaging over all the prediction errors. An extreme form of this type of cross-validation technique is one in which all but one data point is used as the training set and the last one is used as the test data i.e. given N patterns $N-1$ are used for training and 1 for testing. This leads to N such partitions and is called the Leave one out method(LOO). Leave one out is attractive since it uses all the data available for training. Also for linear models like RBFNs, LOO has a closed form equation for variance of the prediction error estimate

$$\hat{\sigma}^2_{LOO} = \frac{\mathbf{t}^T\mathbf{P}(diag(\mathbf{P}))^{-2}\mathbf{P}\mathbf{t}}{\mathbf{P}} \tag{29}$$

where $\mathbf{P}$ is the projection matrix.

5.1.3 Ridge Regression

Regularization involves adding a penalty functional to the cost so as to bias the solution towards more desirable solutions. For RBFNs, a popular choice for the modified cost function is

$$SSE = \sum_{n=1}^{N}\{t^n - y(\mathbf{x}^n)\}^2 + \lambda\sum_{j=1}^{M} w_j^2. \tag{30}$$

The above expression is associated with ridge regression where λ is the regularization parameter and models with large weights at the output layer are penalized. As λ increases the network function becomes smoother. Due to the change in the error function, the variance matrix now becomes

$$\mathbf{A} = \mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I}_M. \tag{31}$$

This matrix is better conditioned than $\mathbf{\Phi}^T\mathbf{\Phi}$, a useful property since the optimal weight vector is now

$$\mathbf{W}^T = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I}_M)^{-1}\mathbf{\Phi}^T\mathbf{T}. \tag{32}$$

The projection matrix is still given by Eq. 28, but using Eq. 31 for $\mathbf{A}$.

The regularization parameter λ can be chosen by minimizing the prediction error as estimated by using K-fold cross validation, generalized cross validation or some other criterion. However one of the most convenient methods is to use the generalized cross validation(GCV) since it leads to the following reasonably simple iterative formula for the estimate of regularization parameter [Orr96]:

$$\hat{\lambda} = \frac{\mathbf{T}^T\mathbf{P}^2\mathbf{T} trace(\mathbf{A}^{-\mathbf{1}} - \hat{\lambda}\mathbf{A}^{-\mathbf{2}})}{\mathbf{W}\mathbf{A}^{-1}\mathbf{W}^T trace(\mathbf{P})}. \tag{33}$$

5.1.4 Local Ridge Regression

This is a variation of ridge regression where each weight term is penalized differently thereby associating a different regularization parameter with each basis function. The error function is given by:

$$E = \sum_{n=1}^{N} \{t^n - y(\mathbf{x}^n)\}^2 + \sum_{j=1}^{M} \lambda_j w_j^2. \tag{34}$$

This is called local ridge regression because the basis functions used have localized responses. Hence the effect of each regularization parameter is to allow the smoothness to be adapted to local conditions. This is especially useful when the underlying function has different smoothness properties in different parts of the input space. The effect of the change in the error function affects the form of the variance matrix as:

$$\mathbf{A}^{-1} = (\mathbf{\Phi}^T\mathbf{\Phi} + \mathbf{\Lambda}) \tag{35}$$

where $\mathbf{\Lambda}$ is a diagonal matrix containing the regularization parameters $\{\lambda\}_{j=1}^{M}$ along its diagonal.

5.2 Pruning and Growing RBFNs

The concept of projection matrix and the associated geometrical interpretation of SSE provides an appealing way of growing an RBFN using forward selection.

5.2.1 Forward Selection

One is given an initial network configuration and a candidate pool of basis functions, typically Gaussians centered at the training data points. At each step, the hidden unit(basis function) which decreases the error(such as sum squared error) most, is removed from the candidate pool and added to the network. This process of adding hidden units and increasing the model complexity is continued till some criterion such as GCV stops decreasing. Though forward selection is a nonlinear optimization technique, it has the advantage of not having to fix the number of units in advance and also a tractable model order selection criterion. Besides it is computationally efficient. The projection matrix for the case where an extra hidden unit has been added is given by

$$\mathbf{P}_{M+1} = \mathbf{P}_M - \frac{\mathbf{P}_M\mathbf{f}_{M+1}\mathbf{f}_{M+1}^T\mathbf{P}_M}{\mathbf{f}_{M+1}^T\mathbf{P}_M\mathbf{f}_{M+1}} \tag{36}$$

where $\mathbf{f}_{M+1}$ is the column of the design matrix $\mathbf{\Phi}$ corresponding to the most recently recruited hidden unit. The reduction in the sum squared error due to the addition of the unit is given by

$$E_M - E_{M+1} = \frac{(\mathbf{t}\mathbf{P}_M\mathbf{f}_{M+1})^2}{\mathbf{f}_{M+1}^T\mathbf{P}_M\mathbf{f}_{M+1}} \tag{37}$$

The unit which reduces the sum squared error most is the chosen candidate. Geometrically this will be the unit whose corresponding basis vector is most closely aligned to the perpendicular from $\mathbf{t}$ to the current M-dimensional space. Though the sum squared error reduces as more hidden units are added, the model selection criterion will reach a minima before it starts increasing again. This is the point where the process is stopped. Forward selection and ridge regression can be combined to yield further improvement in the choice of model complexity. A sophisticated variant of forward selection based on *orthogonal least squares* has been proposed [Orr96].

5.2.2 Backward Elimination

In backward elimination, a network is constructed with all the basis functions in the candidate pool. At each step, the unit which least increases the error, is eliminated from the network. Again, this procedure is continued till some model selection criterion stops decreasing. At this point, the complexity of the model is assumed to be sufficient to represent the underlying function complexity.

5.3 Hierarchical Growing

As discussed in the next section, σ directly relates to the notion of scale in RBFNs. This was exploited by Moody (1989) who proposed a multi-resolution approach to training and growing an RBFN. The network would be first trained using large but fixed values of σ, to form a coarse approximation. The residual error would then be modelled by additional kernels with smaller values of σ. Moody showed that such a hierarchical, coarse-to-fine training is efficient and suited to real-time applications. He used B-splines as activation functions, but also noted that they are not convenient for high-dimensional spaces. Subsequently, the use of wavelet based kernels, recast in the RBFN framework, was advocated as a powerful, multi-resolution approach to function approximation [BS93].

Another interesting approach to network growing for classification problems is presented in [LK91]. With each Gaussian hidden unit is associated a region of influence as well as a class label. As training samples are examined, parameters are adjusted in the usual way unless a sample does not fall in the region of influence of any hidden unit with the same class label. In the latter case, a new hidden unit, positioned on top of that sample, is created with the same class label. Also, if the MSE flattens out at an unacceptable level, the width of each Gaussian is decreased, so that additional kernel units are more likely to be created in the next iteration.

5.3.1 Online approaches: The Resource Allocating Network

The resource allocating network[Pla91] is similar in spirit to the hierarchical technique mentioned above, but works for function approximation problems also and is tailored for on-line learning. Again, the motivation of such an approach is to match the model complexity with that of the data. The resource allocating network uses a sequential learning technique. As new data elements are presented , based on two thresholds viz. a prediction error threshold and a data novelty threshold, it either adapts the existing network parameters or grows the network. Kadirkamanthan and and Niranjan[KN93] provide a theoretical foundation for RANs using a sequential function estimation approach based on the principle of F projection. The principle states that the posterior estimate of an unknown function $f^{(n)}$ can be obtained from its prior estimate $f^{(n-1)}$ and a new observation $I^{(n)}$ such that the function has the least L_2 norm $||f_{(n)} - f_{(n-1)}||$. Using the same approach Molina and Niranjan [MN96] provide a pruning with replacement rule for RAN architectures with limited number of units. Such a rule is useful in an online setting where the network is implemented in hardware. A relevance criteria is associated with each hidden unit. When a new observation arrives, based on the two thresholds mentioned above, it is determined whether a new unit needs to be added. If no more hidden units are available, then the relevance of the new unit is compared with the existing ones. If any unit is found which has a lower relevance, then it is replaced with the new unit. Thus we obtain a pruning rule which helps in increasing the information contained in the network having limited resources. More recently, a pruning algorithm for RAN, that removes units showing little activation over a windowed period, has been presented along with strong comparative experimental results [YSS97].

6 The role of scale in RBFNs

The width parameter, σ, invokes the notion of scale in families of functions, including wavelets. Previously we saw that the chosen scale impacted the degree of smoothness of the function realized by an RBFN. Also, a popular way of obtaining center locations is through clustering. Now clustering naturally involves the *scale* idea since at a sufficiently small scale every sample point becomes a cluster, and at some large scale the entire data set may be viewed as a single cluster. The question then is, can the clustering procedure and the choice of σ be viewed in a unifying framework, and does this provide added insight into what type of function an RBFN realizes? This question is addressed in [Cha96], and the answers are multi-faceted and very positive.

6.1 Training Centroids using Scale-based Clustering

First, it is observed that a gaussian RBFN can by itself cluster data in a *scale-based* manner. In a scale-space approach to clustering, clusters are determined by analyzing the data over a range of scales, and clusters that are stable over a considerable scale interval are accepted as *true* clusters [Wit83], [Won93]. Suppose we assign w_{kj}, t_k^n and σ_j constant values, w, t and σ respectively, with $t > 0$ and $w/t << 1$, Thus we are training an RBFN with a contant output value, and the only variables to be estimated are the center locations. In this setting, eq. 20 can be approximated by:

$$\Delta\boldsymbol{\mu}_j = (\eta_4)\frac{||\mathbf{x}^n - \boldsymbol{\mu}_j||}{\sigma_j^2} e^{\left(-\frac{||\mathbf{x}^n - \boldsymbol{\mu}_j||^2}{2\sigma_j^2}\right)}, \tag{38}$$

where $\eta_4 = \eta_2 w$. At equilibrium one obtains:

$$\nabla(\phi_j(\mathbf{x};\sigma) * p(\mathbf{x})) = 0. \tag{39}$$

where $p(\mathbf{x})$ is the input probability density and '*' is the convolution operator. Centroids obtained by iterating eqn. (38) are the extrema of the input density $p(\mathbf{x})$, smoothed by a gaussian of width σ. Therefore, the width acts as a scale parameter for clustering; the number of clusters obtained is a function of the width, and decreases as width is increased.

Eq. 38 can be used for clustering problems. On the well-known Iris classification data, the above method outcompeted standard clustering algorithms like FORGY and CLUSTER even though it did not use class information [CG96].

The right set or number of clusters, in other words the right width value, can be picked by computing cluster centers for a range of width values. Clusters that hardly change over a long range of width values (using a logarithmic scale), are natural clusters for the given data, and the centers of these clusters are the solutions to Eq. 39. Thus this procedure, in one shot, provides center locations, width values as well as model selection in terms of the number of centers needed.

6.2 Weight Training and Network Complexity

An important problem that is encountered in training of a feedforward network is: how to match the network complexity (size, number of trainable parameters etc.) with the problem complexity.

We addressed some aspects of this issue while studying model selection for RBFNs. On contrasting the localized fitting of RBFNs with spline models which generate piece-wise, smooth fit to observational data [Wah90], one can identify cases where the localized receptive field approach places increased demands on the network complexity for fitting even simple problems. For example, consider the task of learning a low-order polynomial relation, $y = \sum_{i=0}^{n-1} a_i x^i$, using a RBFN over a finite domain. Although the target function complexity is described by the number of parameters (n), in a local approximation approach, the required RBFN size increases with the size of the input domain. The desired network complexity is unrelated to target complexity. This problem can be mended by getting rid of local approximation approach to training RBFN.

The usual path (of centroid clustering followed by weight matrix inversion) for RBFN training is meaningful only for small width values. When widths are allowed to be large, the localized receptive field concept becomes inapplicable. In that new regime, the width parameter controls the network complexity independent of network size. To take an extreme situation, consider what happens when widths of an RBFN are very large compared to spread of the sample data. The network output stays close to a constant value independent of the input vector and the number of hidden units. That is, width determines complexity independent of the network size.

More generally width can be used to control the nonlinearity of network function. For simplicity, consider the output of a single-input, single-output RBFN given by,

$$y(x) = \sum_{j=0}^{N} w_j \phi_j(x; \sigma_j) \tag{40}$$

Assuming uniform width ($\sigma_j = \sigma, \forall j$), and Gaussian RBFs the output can be expressed as,

$$y(x) = \sum_{j=0}^{n} w_j \left(1 - \frac{(x-\mu_j)^2}{\sigma^2} + ...\right)$$

or,

$$y(x) = L_m(x; w_j; \sigma) + H_m(x; w_j; \sigma) \tag{41}$$

where, L_m comprises of the m lower-order terms in x, and H_m includes the remaining higher-order terms. Note that a higher value of σ, the network output $y(x)$ approximates a polynomial of degree $m-1$ with error,

$$\|y(x) - L_m(x; w_j; \sigma)\| \leq O(\sigma^{-2m}). \tag{42}$$

Further, it can be easily verified that coefficients of the polynomial L_m linearly depend on network weights, w_j. Therefore, in principle, by proper choice of σ and weights, w, the RBFN can approximate a mth degree polynomial with only $m+1$ RBF nodes. In this way the target complexity could be matched with network complexity, with the width parameter playing a crucial role in this process [HLC93].

The procedure to train RBFN so as to approximate polynomials is distinct from the usual training procedures since it does not create a local approximation of the target. The steps in the training process are given below [CG97]:

- Step 1: Choose a RBFN with $m+1$ RBF nodes to fit a univariate mth degree target polynomial. (The case of multi-variate targets is much more complicated. Some special results are derived using the tools of Catastrophe Theory in [CG97]). Also choose a uniform width value ($\sigma_j = \sigma, \forall j$).

- Step 2: Train the centroids using a standard clustering procedure, or using Eq. 38 of previous section.

- Step 3: The weights are calculated by minimizing squared output error. This is must be done by a pseudoinversion procedure like the Singular Value Decomposition [GL89].
- Step 4: Calculate the squared output error, E, obtained after training the network.
- Step 5: Update σ as: $\sigma \leftarrow \beta\sigma$, where $\beta > 1$. If $\sigma > \sigma_{max}$ terminate, else go back to Step 3.

The desired polynomial approximation is obtained at a value of σ that globally minimizes $E(\sigma)$.

The above polynomial approximation scheme has been used to calculate a local fit to a smooth target function depending on the local "shape" of the target. According to Catastrophe theory (CT) ([Tho75],[Gil81]), the shape of a smooth function, $f(\mathbf{x})$, is determined by its Critical Points (CP) ($\mathbf{x}_0$s), defined as points where $\nabla f(\mathbf{x}) = 0$. Depending on the Hessian at $\mathbf{x}_0$, CT puts each CP into one of several distinct types. The complexity of $f(\mathbf{x})$ near a CP is described by a parameter known as the *codimension*. In [CG97], it has been shown how a smooth target of a known "shape" can be locally approximated using a RBFN, such that the desired network complexity is related to codimension, i.e. the target complexity.

In summary, we note that the goal of matching target complexity to network complexity is frustrated by the traditional local approach to approximation using RBFN. This difficulty is overcome by allowing very large width values. This takes us into a new regime of RBFN parameter space where the network can be made to approximate polynomial targets accurately. It is then possible to design a training scheme where the desired network size is closely related to the target complexity. The design of RBFNs where different basis functions may have widely varying width values promises to be an exciting area of investigation.

7 Normalized RBFNs

By adding a normalizing factor to the equation of the basis function, one obtains normalized radial basis functions :

$$\phi_i(\mathbf{x}) = \frac{\phi(||\mathbf{x} - \boldsymbol{\mu}_i||)}{\sum_{j=1}^{M} \phi(||\mathbf{x} - \boldsymbol{\mu}_j||)} \tag{43}$$

where M is the total number of kernels. A network obtained using the above form for the basis functions is called a normalized RBFN and has several interesting properties. Firstly this is the natural form of the function obtained in various settings such as noisy data interpolation and regression [Bis95]. Also since the basis function activations are bounded between 0 and 1, they can be interpreted as probability values, especially in classification applications as will be seen below.

7.1 Classification using radial basis function networks

Radial basis function networks can be effectively used for classification purposes. We will look at two ways of interpreting the classification capabilities of RBFNs. Both interpretations can be obtained by comparing the approach to a Bayesian classification strategy. Consider the data set as shown in Fig. 2. The data points belonging to the different classes can either be separated using hyperplanes(as in multilayered perceptrons) or alternatively by placing localized kernels around each group. The latter approach is associated with RBFNs. One could first model the class conditional densities for each class. Using these densities and appropriate class priors, Bayes theorem could be applied to obtain the posterior probabilities of each class. The classification rule then consists of choosing the class with the highest posterior probability. Once can model this with an RBFN in which, each basis function models the class conditional density for each class. The k^{th} hidden unit is only connected to the k^{th} output unit to represent the corresponding prior. Thus the k^{th} output of the overall network can be viewed as the posterior probability $P(C_k|\mathbf{x})$ of the pattern $\mathbf{x}$ belonging to class C_k :

$$P(C_k|\mathbf{x}) = P(C_k)\phi_k(\mathbf{x}) \tag{44}$$

where

$$\phi_k(\mathbf{x}) = \frac{p(\mathbf{x}|C_k)}{\sum_i p(\mathbf{x}|C_i)P(C_i)} \tag{45}$$

and $p(\mathbf{x}|C_k)$ is the class conditional density.

Instead of assigning a separate basis function for each class conditional density, a mixture model could be used for modelling the densities. An efficient method involves the use of a common set of M basis functions to represent all class conditional distributions. This leads to a normalized RBFN:

$$P(C_k|\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}). \tag{46}$$

where

$$\phi_j(\mathbf{x}) = \frac{p(\mathbf{x}|j)P(j)}{\sum_{i=1}^{M} p(\mathbf{x}|i)P(i)} = P(j|\mathbf{x}) \tag{47}$$

and the hidden to output unit weights are given by

$$w_{kj} = \frac{P(j|C_k)P(C_k)}{P(j)} = P(C_k|j) \tag{48}$$

In this scenario, the hidden unit centers can be considered as representative feature vectors and hidden unit activations can be interpreted as the posterior probabilities of the presence of corresponding features in the input space. Thus the overall output of the network represents the posterior probability of a class membership, given these features. The weights provide the prior probability of each feature.

7.2 Noisy data interpolation theory

Consider a mapping from a one-dimensional input x to a one-dimensional output y such that the output data is obtained from a smooth noise-free function $h(x)$. However assume that the input has been corrupted by additive noise. Then the sum-of-squares error will be given by

$$E = \frac{1}{2} \int \int \{y(x + \zeta) - h(x)\}^2 q(\zeta) p(x) d\zeta dx \tag{49}$$

where $p(x)$ and $q(\zeta)$ are the probability density functions of the input data and noise respectively. If we assume the distribution of noise to be normal i.e. $q(\zeta) \propto exp(-\zeta^2/2\sigma^2)$, then minimization of the error term for a finite training data set $\{x^n, t^n\}$ gives:

$$y_k(x) = \frac{\Sigma_n h_k(x^n) exp\{-(x - x^n)^2/2\sigma^2\}}{\Sigma_n exp\{-(x - x^n)^2/2\sigma^2\}}. \tag{50}$$

This expression can be looked upon as an expansion in normalized radial basis functions with the weights given by $h(x^n)$ and the basis functions given by

$$\phi(x - x^n) = \frac{exp\{-(x - x^n)^2/2\sigma^2\}}{\Sigma_n exp\{-(x - x^n)^2/2\sigma^2\}}. \tag{51}$$

The weights $h(x^n)$ are unknown and have to be determined from the training data points. However we know that the function $h(x)$ is noise-free and hence we have $h(x^n) = t^n$. Thus noisy data interpolation theory provides another insight into the origin of radial basis function expansions.

7.3 Kernel Regression

The theory of kernel regression provides another viewpoint to the use of radial basis functions for function approximation. It provides a framework for estimating regression functions for noisy data using kernel density estimation techniques. Let us reconsider the problem of mapping a d-dimensional input vector $\mathbf{x}$ to an output vector $\mathbf{y}$ of dimensionality c. Suppose we are given a set of training data of N data samples $\{\mathbf{x}^n, \mathbf{t}^n\}$. The joint probability density $p(\mathbf{x}, \mathbf{t})$ can be modelled using a Parzen kernel estimator. If we use Gaussian kernel functions, then the estimator will have the form

$$\hat{p}(\mathbf{x}, \mathbf{t}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{(2\pi h^2)^{(d+c)/2}} e^{\left(-\frac{||\mathbf{x}-\mathbf{x}^n||^2}{2h^2} - \frac{||\mathbf{t}-\mathbf{t}^n||^2}{2h^2}\right)}. \tag{52}$$

The objective of function approximation is to find a mapping from the input space to the output space. Under many circumstances, it can be shown that the mapping is provided by forming the regression, or conditional average of the target data, conditioned on the input variables. The conditional average can be expressed in terms of the conditional density which in turn can be derived from Eq. 52. The regression function is known as the Nadaraya-Watson estimator [Nad64, Wat64] and is given by :

$$\mathbf{y}(\mathbf{x}) = \frac{\sum_n \mathbf{t}^n e^{\left(-\frac{||\mathbf{x}-\mathbf{x}^n||^2}{2h^2}\right)}}{\sum_n e^{\left(-\frac{||\mathbf{x}-\mathbf{x}^n||^2}{2h^2}\right)}} \tag{53}$$

It has been rediscovered in the context of neural networks[Spe90, SH92]. The above expression can be looked upon as an expansion in normalized radial basis functions with the expansion functions once again being provided by the target data $\mathbf{t}_n$. As discussed earlier, assigning a kernel for each data point greatly increases the cost of computing. Thus the joint probability distribution could be modelled using an adaptive mixture model instead of the kernel estimator. Algorithms such as the EM(expectation-maximization) algorithm could then be used for determining the parameters of the model.

7.4 Solution for missing variables

Many real world applications need to deal with data having incomplete feature vectors or missing variables. Image occlusion and sensor failure are examples of cases which present the need to find efficient methods of training and prediction in the presence of incomplete data. The use of heuristic methods, such as substituting the mean value for the unknown feature, lead to sub-optimal solutions and in some cases may be worse than neglecting the sample. It has been shown [AT93, TN94, TA95] that the optimal solution involves integrating the network output over the missing dimensions weighted by local probability densities. Efficient closed form solutions exist for Gaussian basis function networks. [TA95] showed that the use of Parzen windows with Gaussian kernels for estimating probability densities leads to a solution in the form of an RBF network of normalized Gaussians centered at the known components of data points. Assume $NN(\cdot)$ is a neural network which has been trained to predict $E(y|\mathbf{x})$ where $\mathbf{x}$ is a d-dimensional vector and y is a 1-dimensional scalar. If such a network is presented with an incomplete input vector $\mathbf{x} = (\mathbf{x}^c, \mathbf{x}^u)$ where $\mathbf{x}^c$ denotes the known inputs and $\mathbf{x}^u$ denotes the unknown inputs, the optimal prediction for $E(y|\mathbf{x}^c)$ would be given by:

$$E(y|\mathbf{x}^c) \approx \frac{\sum_{k=1}^{N} \alpha_k G(\mathbf{x}^c; \mathbf{x}^{c,k}, \sigma)}{\sum_{k=1}^{N} G(\mathbf{x}^c; \mathbf{x}^{c,k}, \sigma)}, \quad \alpha_k = NN(\mathbf{x}^c, \mathbf{x}^{u,k}) \tag{54}$$

under the condition that the input probability density is estimated using Parzen windows with Gaussian kernels. Here k iterates over N training points, $NN(\mathbf{x}^c, \mathbf{x}^{u,k})$ is the neural network prediction obtained by substituting the corresponding components of the training data points for the unknown input and $G(\mathbf{x}^c; \mathbf{x}^{c,k}, \sigma)$ is a Gaussian kernel whose center $\mathbf{x}^{c,k}$ is obtained by considering only those components which correspond to the known input. Ghahramani and Jordan [GJ94] provide an EM approach to supervised learning using incomplete data where the joint probability density of the data is modelled using a mixture of Gaussian basis functions.

8 Applications

Since RBFNs have powerful function approximation capabilities, good local structure and efficient training algorithms, they have been used for a wide variety of applications such as chaotic time series prediction, speech pattern classification, image processing, medical diagnosis, nonlinear system identification, adaptive equalization in communication systems, nonlinear feature extraction etc. [Hay94, Mul96, Web96]. The various applications described in this book, including hand gesture recognition, face recognition, classification of ECGs, mapping hand gestures to speech, object recognition and minefield detection, all involve an underlying mapping problem that exploits the universal and smooth function approximation capabilities of RBFNs.

Since in many of these applications, the RBF is used as a static function approximator, the question arises, why not use an MLP instead? The suitability of course depends on the nature of the problem. For several applications, the two types of networks, if chosen and trained carefully, are seen to attain similar levels of performance [Lip91]. Often an RBF is quicker to train but is more memory intensive as compared to an MLP. Also, as with any kernel based approach, it is more susceptible to the curse of dimensionality [Fri94].

But for many applications, the RBFN is a natural choice, and adds extra capabilities. For example, the error observed for a training sample can be apportioned among the localized hidden units depending on their proximity. Then, for a new sample, the output can be supplemented by a weighted average of the "errors" or uncertainty associated with each kernel unit that contributed to that output. In this way, input-dependent confidence intervals can be obtained for the output estimate [LKU92].

The link between Bayesian classifiers and the normalized RBFN has been shown in Sec. 7.1. This relationship is specifically exploited in applications such as design of adaptive equalizers [Mul96]. Some other applications are based on relating RBFNs to fuzzy inference systems [JS93, SC95, PM92]. Here the kernels are viewed as providing membership functions, whose outputs pass through one or more layers of nodes for fuzzy inferencing. Moreover, the backpropagation algorithm can be extended to modify the kernel location and widths, and thereby the membership functions [SC95, TG97].

Like the MLP, the RBF network can be readily extended for dealing with dynamic maps. In signal processing applications such as speech processing and forecasting, a tapped delay line has been used to feed the current and past input values into an RBF network [Mul96]. Recurrent connections reminiscent of the IIR filter have also been proposed. A nice application to face and gesture recognition incorporates past information into the input vector by using attenuated feedback from the previous state of the input vector [RYD96]. RBFNs are expected to continue to enjoy popularity as a versatile and practical nonlinear function approximator for a variety of engineering applications.

Acknowledgements

This research was supported in part by the NSF under Grant ECS-9900353, and ARO under contract DAAG55-98-1-0230.

References

[AT93] S. Ahmad and V. Tresp. Some solutions to missing feature problem in vision. In Jack D. Cowan Stephen J. Hanson and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 393–400. The MIT Press, 1993.

[BFG95] M. Bianchini, P. Frasconi, and M. Gori. Learning without local minima in radial basis function networks. *IEEE Transactions on Neural Networks*, 6(3):749–756, 1995.

[BG92] S. Beck and J. Ghosh. Noise sensitivity of static neural classifiers. In *SPIE Conf. on Applications of Artificial Neural Networks SPIE Proc. Vol. 1709*, pages 770–779, Orlando, Fl., April 1992.

[Bis95] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, New York, 1995.

[BL88] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[BP96] A. G. Borş and I. Pitas. Median radial basis function neural network. *IEEE Transactions on Neural Networks*, 7(6):1351–1364, November 1996.

[BS93] Bhavik R. Bakshi and George Stephanopoulos. Wave-net: a multiresolution, hierarchical neural network with localized learning. *AIChE Journal*, Vol.39, No.1:57–81, January 1993.

[CDS90] Yann Le Cun, J.S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems-2*, pages 598–605, 1990.

[CG96] S. V. Chakaravathy and J. Ghosh. Scale based clustering using a radial basis function network. *IEEE Transactions on Neural Networks*, 5(2):1250–61, Sept 1996.

[CG97] S. V. Chakaravathy and J. Ghosh. Function emulation using radial basis function networks. *Neural Networks*, 10:459–478, May 1997.

[Cha96] S.V. Chakravarthy. *On the role of singularities in neural networks.* PhD thesis, Dept. of Elect. Comp. Eng., Univ of Texas, Austin, May, 1996.

[Fri94] J. H. Friedman. An overview of predictive learning and function approximation. In V. Cherkassky, J.H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks, Proc. NATO/ASI Workshop*, pages 1–61. Springer Verlag, 1994.

[GBD92] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[GC94] J. Ghosh and S.V. Chakravarthy. The rapid kernel classifier: A link between the self-organizing feature map and the radial basis function network. *Jl. of Intelligent Material Systems and Structures*, 5:211–219, 2 1994.

[GDB92] J. Ghosh, L. Deuser, and S. Beck. A neural network based hybrid system for detection, characterization and classification of short-duration oceanic signals. *IEEE Jl. of Ocean Engineering*, 17(4):351–363, October 1992.

[Gil81] R. Gilmore. *Catastrophe Theory for Scientists and Engineers*. Wiley Interscience, New York, 1981.

[GJ94] Z. Ghahramani and M. Jordan. Supervised learning from incomplete data via an em approach. In Gerald Tesauro Jack D. Cowan and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 120–127. The MIT Press, 1994.

[GL89] G. Golub and C.Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, MD., 1989.

[GT94] J. Ghosh and K. Tumer. Structural adaptation and generalization in supervised feedforward networks. *Jl. of Artificial Neural Networks*, 1(4):431–458, 1994.

[Hay94] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.

[HLC93] J. C. Hoskins, P. Lee, and S. V. Chakravarthy. Polynomial modelling behavior in radial basis function networks. In *Proc. of World Congress on Neural Networks*, pages 693–699, Portland, OR, July 1993.

[JS93] J. S. Roger Jang and C. T. Sun. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks*, 4(1):156–159, January 1993.

[KH92] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In S.J. Hanson J.E. Moody and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems-4*, pages 950–957. Morgan Kaufmann, San Mateo, CA, 1992.

[KHK90] J. Kowalski, E. Hartman, and J. Keeler. Layered neural networks with gaussian hidden units as universal approximators. *Neural Computation*, 2:210–215, 1990.

[KL97] Adam Krzyżak and Tamás Linder. Radial basis function networks and complexity regularization in function learning. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 197, 1997.

[KLL96] A. Krzyzak, T. Linder, and G. Lugosi. Nonparametric estimation and classification using radial basis function nets and empirical risk minimization. *IEEE Transactions on Neural Networks*, 7(2):475–487, March 1996.

[KN93] V. Kadirkamanathan and M. Niranjan. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5:954–975, 1993.

[Lig92] W. A. Light. Some aspects of radial basis function approximation. In S.P. Singh, editor, *Approximation Theory, Spline Functions and Applications*, pages 163–90. NATO ASI Series Vol. 256, Kluwer Acad., Boston, 1992.

[Lip91] R. P. Lippmann. A critical overview of neural network pattern classifiers. In *IEEE Workshop on Neural Networks for Signal Processing*, 1991.

[LK88] Sukhan Lee and Rhee M. Kil. Multilayer feedforward potential function network. In *Proceedings of the Second International Conference on Neural Networks*, pages 161–171, 1988.

[LK91] Sukhan Lee and Rhee M. Kil. A gaussian potential function network with hierarchical self-organizing learning. *Neural Networks*, 4:207–224, 1991.

[LKU92] J.A. Leonard, M.A. Kramer, and L.H. Ungar. Using radial basis functions to approximate a function and its error bounds. *IEEE Transactions on Neural Networks*, 3:4:624–627, July 1992.

[Mac95] D. J. C. Mackay. Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995.

[MD89] J. Moody and C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[Meg61] P. Megdassy. *Decomposition of superposition of distributed functions.* Hungarian Academy of Sciences, Budapest, 1961.

[Mic86] C.A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

[MN96] C. Molina and M. Niranjan. Pruning with replacement on limited resource allocating networks by F-projections. *Neural Computation*, 8:855–868, 1996.

[Moo94] J.E. Moody. Prediction risk and architecture selection for neural networks. In V. Cherkassky, J.H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks, Proc. NATO/ASI Workshop*, pages 143–156. Springer Verlag, 1994.

[Mul96] B. Mulgrew. Applying radial basis functions. *IEEE Signal Processing Magazine*, pages 50–65, March 1996.

[Nad64] E. A. Nadaraya. On estimating regression. *Theory of Probability and its Applications*, 9 (1):141–142, 1964.

[NG96] P. Niyogi and F. Girosi. On the relationship between generalization error, hypothesis complexity and sample complexity for radial basis functions. *Neural Computation*, 8:819–842, 1996.

[OF94] Hossam Osman and Moustafa M. Fahmy. Probabilistic winner-take-all learning algorithm for radial-basis-function neural classifiers. *Neural Computation*, 6(5):927–943, 1994.

[Orr95] M.J.L. Orr. Regularization in the selection of radial basis function centers. *Neural Computation*, 7:606–623, 1995.

[Orr96] M. J. L. Orr. Introduction to radial basis function networks. Technical Report April, Center for Cognitive Science, Univ. of Edinburgh, 1996.

[PG90] T. Poggio and F. Girosi. Networks for approximation and learning. *Proc. IEEE*, 78(9):1481–97, Sept 1990.

[Pla91] John C. Platt. A resource allocation network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[PM92] Sankar K. Pal and Sushmita Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on Neural Networks*, 3(5):683–697, September 1992.

[Pow85] M.J.D. Powell. Radial basis functions for multivariable interpolation: A review. In *IMA Conf. on Algorithms for the approximation of functions and data*, pages 143–167, 1985.

[PS91] J. Park and I.W. Sandberg. Universal approximation using radial basis function networks. *Neural Computation*, 3(2):246–257, Summer 1991.

[PS93] J. Park and I.W. Sandberg. Universal approximation and radial basis function networks. *Neural Computation*, 5(??):305–316, 1993.

[Ree93] R. Reed. Pruning algorithms–a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, September 1993.

[RYD96] M. Rosenblum, Y. Yacoob, and L. S. Davis. Human expression recognition from motion using a radial basis function network architecture. *IEEE Transactions on Neural Networks*, 7(5):1121–1138, September 1996.

[SC95] C. Shim and J.Y. Cheung. Pattern classification using RBF based fuzzy neural network. In *Intelligent Engineering Systems Through Artificial Neural Networks*, volume 5, pages 485–90. ASME Press, November 1995.

[SH92] H. Schioler and U. Hartmann. Mapping neural network derived from the parzen window estimator. *Neural Networks*, 5:903–909, 1992.

[Spe90] J. Specht. Probabilistic neural networks. *Neural Networks*, 3:45–74, 1990.

[TA95] R. Tresp, V. Neuneier and S. Ahmad. Efficient methods for dealing with missing data in supervised learning. In David S. Touretzky Gerald Tesauro and Todd K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 687–696. The MIT Press, 1995.

[TG97] I. Taha and J. Ghosh. Hybrid intelligent architecture and its application to water reservoir control. *International Journal of Smart Engineering Systems*, 1:59–75, 1 1997.

[Tho75] R. Thom. *Structural Stability and Morphogenesis.* Reading:Benjamin, 1975.

[TN94] S. Tresp, V. Ahmad and R. Neuneier. Training neural networks with deficient data. In Gerald Tesauro Jack D. Cowan and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 128–135. The MIT Press, 1994.

[Wah90] G. Wahba. *Spline Models for Observational Data*, volume 59. Society for Industrial and Applied Mathematics, Philadephia, 1990.

[Wat64] G. S. Watson. Smooth regression analysis. *Sankhya: The Indian Journal of Statistics*, Series A 26:359–372, 1964.

[Web96] A. R. Webb. An approach to non-linear principal component analysis using radially symmetric kernel functions. *Statistics and Computing*, 9:159–68, 1996.

[Wit83] A. P. Witkin. Scale-space filtering. In *8th Int. Joint Conf. Art. Intell.*, pages 1019–1022, Karlsruhe, Germany, August 1983.

[Won93] Y. F. Wong. Clustering data by melting. *Neural Computation*, 5(1):89–104, 1993.

[WS96] A. R. Webb and S.M. Shannon. Spherically-symmetric basis functions for discrimination and regression: determining the nonlinearity. Technical Report 651/2/JP95/22, DRA/CIS, Malvern, England, 1996.

[XKY94] L. Xu, A. Krzyzak, and A. Yuille. On radial basis function nets and kernel regression: Statistical consistency, convergence rates and receptive field size. *Neural Networks*, 7:4:609–628, 1994.

[YSS97] L. Yingwei, N. Sundererarajan, and P. Saratchandran. A sequential learning scheme for function approximation using minimal radial basis function networks. *Neural Computation*, 9:461–78, 1997.

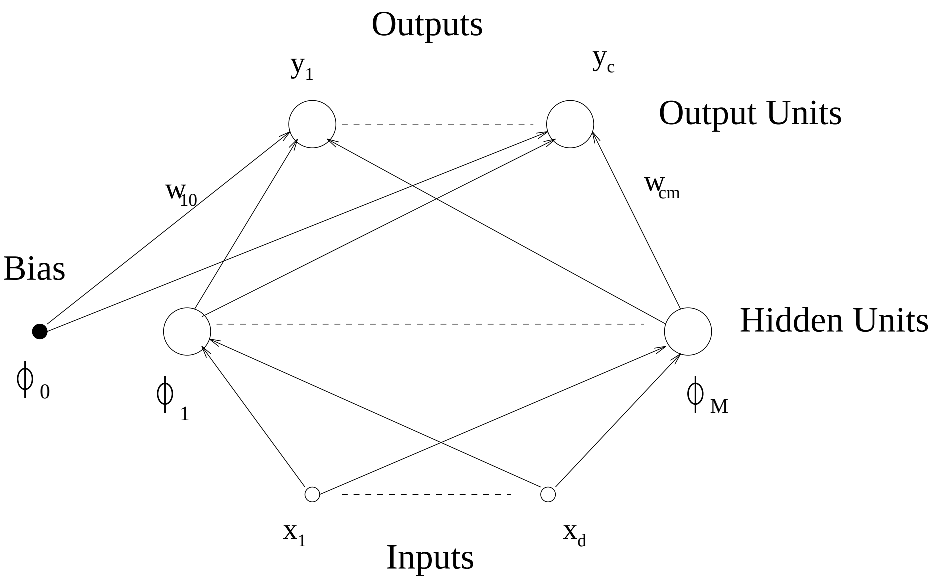

Figure 1: Radial basis function network.

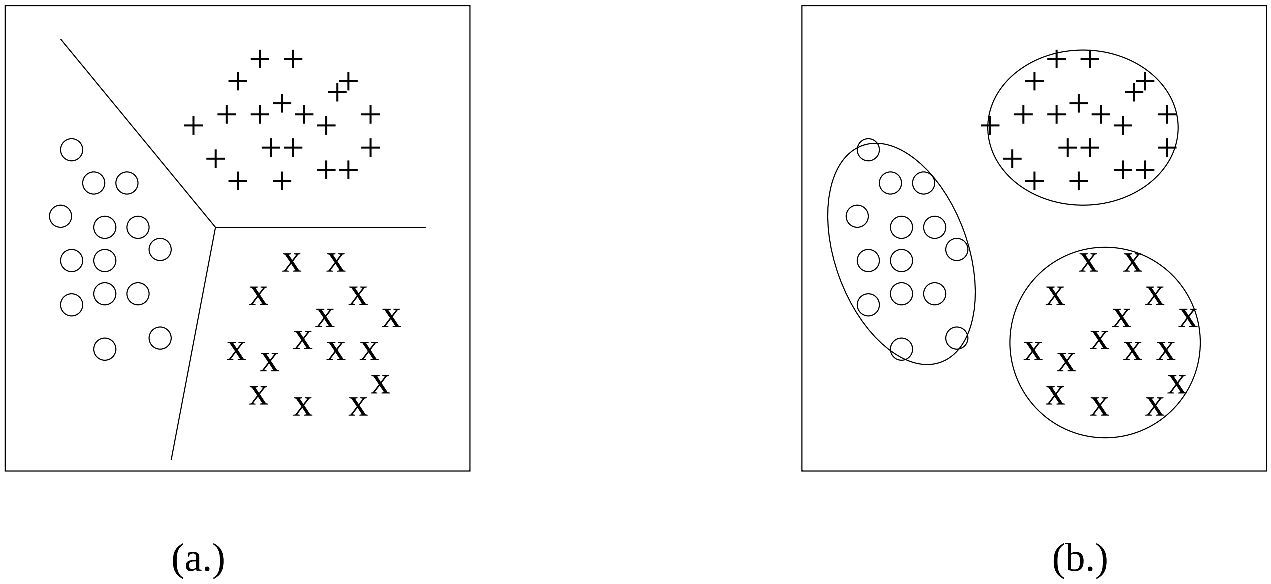

Figure 2: Classification. The 3 different kinds of symbols represent elements of three distinct classes. a] Classes separated using hyperplanes. b] Classes separated using localized kernels.