A Closer Look at the Radial Basis Function (RBF) Networks

K. Mike Tao
Integrated Systems, Inc.
3260 Jay Street, Santa Clara, CA 95054

ABSTRACT

This paper takes a closer look at the increasingly popular RBF networks by showing that not all the RBF networks are the same. They differ in training, in architecture and in the type of RBF used, and consequently they differ in performance and characteristics. Some of them are remarkably better than the others. With analysis and examples, this paper examines the issues of generalization, smoothness of interpolation, and compares different training methods. A robust modelling procedure and a novel fine-tuning procedure are among the recommended features for consistently good performance. Connections with *fuzzy* information processing and with *spline* interpolation are also discussed.

1 Introduction

The conventional multi-layer perceptron (MP) type of feedforward neural network has drawbacks in three aspects. First, the MP architecture tends to over generalize. This is evident in the case of pattern classification. An MP network can be trained to have high accuracy in classifying patterns from a set of known categories, but will also classify any out-of-category pattern as one of the trained categories [1]. This can cause severe problems in real-world applications. Second, the widely used backpropagation (BP) training method is often too slow, especially for large-scale problems (even with improved BP). Third, usually the knowledge represented in an unstructured MP network is not easy to comprehend.

The RBF networks, on the other hand, seem to solve these three problems. First, the most popular RBF network with Gaussian *kernels* learns the pattern probability density functions instead of dividing up the pattern space as the MP networks do. Therefore, when faced with an out-of-category pattern, such an RBF network would be likely to classify it as an unknown category [1]. Second, an RBF network is generally much easier to train than an MP network. This is because the RBF networks establish the RBF parameters directly from the data, and training is primarily on the output layer linear parameters which can be efficiently accomplished by a linear least-squares type of procedure [2]. Third, as shown below, with a *normalization* architecture, the RBF network naturally introduces the notion of *class membership* which allows *fuzzy* partitioning of the input space and a fuzzy rule-based interpretation of the represented knowledge.

However, the RBF networks are not without their own problems. Depending on the architecture, the type of RBF bpernels, and the training methods used, the performance varies. With analysis and examples, this paper examines the issues of generalization, smoothness of interpolation, and compares different training methods. *Normalization* is always recommended, for smooth interpolation and meaningful extrapolation (Section 2). Normalization also offers a (smoothed) *piecewise constant* interpretation of the RBF network. Uneven distribution of training data, however, may contribute to nonsmooth interpolation. In that regard, it is shown in Section 3 that the *thin-plate spline* kernel is less vulnerable than the Gaussian kernel.

To encourage smooth surface learning and reduce the network size, one can use only a subset of the training samples as the RBF kernels. Various kernel selection and training methods are compared in Section 4. It is shown that a *robust, systematic* kernel selection procedure, known as the Orthogonal Least Squares (OLS), performs quite well on a rather difficult response surface learning task. Random seeding, on the other hand, coupled with clustering, may be adequate for simple surfaces. For more difficult surfaces, it is shown that a novel algorithm based on backpropagation (BP) significantly enhances the RBF network's surface learning capability. Finally, in Section 5, connections with *fuzzy inference systems* and extension to *piecewsie linear* RBF networks are discussed.

1058-6393/93 $03.00

2 RBF Network and Generalization

The RBF network accomplishes an input-output nonlinear mapping by linearly combining the nonlinearly transformed inputs according to the following:

$$y = \sum_{i=1}^{m} w_i \phi_i(x) \quad (1)$$

where x is the input vector, y the output vector, and w_i are the output linear combining weights. The $\phi_i()$ are ***radially symmetric*** nonlinear basis functions, hence the term Radial Basis Functions (RBF), and m is the number of RBFs. Multivariate Gaussian, one of the most used RBF ***kernels***, is considered in this section. Other types of RBF kernels are discussed in Section 3.

The RBF transform can be viewed as a network shown in Figure 1, with the RBF kernels represented as "neurons" in the "hidden layer." The network output is simply a linear combination of the hidden neuron outputs. (The "optional" *normalization* layer will be discussed later.)

A typical procedure for *constructing* a RBF network is the following: 1) Use the (input) training data as the centers of RBF kernels. 2) For Gaussian kernels, specify the "standard deviation," σ (by experimenting with the data set for the best results, or by using some sort of averaged "inter-kernel" distance). 3) Find the optimal linear combining weights, w_i, by solving the least-squares (LS) problem defined by Eq. (1).

If all the (input) training samples are *memorized* in the RBF kernels, the LS problem is exactly determined, and the RBF network can learn the *exact* input-output mappings found in the training samples. The question is: can it *interpolate* smoothly (between samples)? The answer to this question will have to be conditioned upon a few factors.

Figure 2 shows an example where the original response surface is sampled at 16 points evenly on a rectangular grid system and is given to a basic RBF network to learn. The basic RBF network has learned the 16 samples exactly, but the interpolated surface is not very smooth as Figure 2 shows. In this example, the σ used for each Gaussian kernel is based on the averaged distance between the underlying kernel and the centers of its two nearest kernel neighbors. We will refer to this σ as the *basic* σ, as it provides basic coverage for the data points. (If we use 5 times the basic σ, the interpolated surface becomes quite smooth. This indicates the sensitivity of smoothness to the choice of σ.)

Another problem with this basic RBF network is that when faced with inputs distant from the training set, the outputs will become very small, if the Gaussian kernels are used. This inability to extrapolate is evidenced in Figure 2. This is because the RBF network is based on memory and familiarity. When faced with unfamiliar cases, it will not respond with vigor. (This RBF-unique feature, however, has the advantage of naturally producing a *confidence measure*.) If pattern classification is of interest, this means that the basic RBF network may classify too many samples as unknown, especially when the training set does not span the pattern space adequately.

In practical applications, sometimes it is desirable to have some "educated guesses." This desire can be realized by using the *normalization* option. The equations then become:

$$y = \sum_{i=1}^{m} (w_i \phi_i(x)) / sum(x) \quad (2)$$

$$sum(x) = \sum_{i=1}^{m} \phi_i(x) \quad (3)$$

With normalization, the RBF network will be able to respond "lively" to unfamiliar cases, i.e., cases distant from the training samples, while still providing the desired *relative confidence measure* which is in $sum(x)$. The question is what kind of answers would it respond with? It will respond in a way *similar* to those cases bearing the most resemblance that one can find in the training set. That is, a normalized RBF network *generalizes* (i.e., both interpolates and extrapolates) according to a *smoothed* nearest-neighbor principle. This becomes clear by using the Gaussian kernel for illustration. If σ is allowed to shrink to 0, it can be readily shown that the normalized RBF performs the nearest-neighbor mapping [2] (and w_i is simply y_i, output of the i-th training sample). The fact that σ is not reduced to zero will make the nearest-neighbor classification *fuzzier*, i.e., multiple "nearest-neighbors" will contribute to the result, thereby providing averaging and smoothness. An example is shown in Figure 3 that with 1/10 of the basic σ, the learned surface clearly shows the nearest-neighbor and *piecewise constant* characteristics. (Generalizing to piecewise linear is discussed later.) With a "nominal" basic σ, the learned surface becomes smoother. And notice that it is much smoother than the one obtained without normalization! Also, extrapolation is carried out in a sensible, nearest-neighbor manner as indicated in Figure 3. With σ increased 10 times further, the normalized RBF network can actually extrapolate the "trend" quite well! Thus, one may use σ as a scale factor for achieving varied resolution effects.

In summary, *normalization* enables the RBF network to extrapolate in a sensible manner, interpolate more

smoothly, and depend less on the choice of σ. In the RBF literature, normalization is usually offered as an "option" or not mentioned [2, 1, 3]. From a practical point of view, it is argued here that normalization should be imperative with few exceptions.

3 Other RBF Kernels

Even with normalization, RBF network's ability to smoothly interpolate can depend on the distribution of the training samples. Figure 4 shows a (hypothetical) response surface displayed with 41 x 41 points. When evenly sampled for 11 x 11 points, a normalized RBF network can interpolate (almost) as identically well as a two-dimensional (cubic) spline.

When sampled *randomly* for 121 points, however, even with normalization, the learned surface is not necessarily smooth (Figure 5). There are (at least) three ways to obtain a smooth surface. One is to use larger σ. However, since the true surface is not known, one has to look for "optimal" σ on test sets generated from the training data. Besides, too large a σ can cause numerical conditioning problems. The second approach is to use only a portion of the training samples as the RBF kernels and this is the subject of Section 4. The third option is to use other types of RBF kernels which would behave differently. The *thin-plate spline* kernel:

$$r^2 log(r) \tag{4}$$

and the Hardy's multi-quadrics are favored by the numerical analysis community [3] for their smooth intrpolation capability (where r is the distance between the input pattern and the kernel's center). We demonstrate the use of *thin-plate spline* RBF kernel in Figure 5, with exactly the same 11 x 11 random points. The learned surface is much smoother. It is almost a little too smooth and becoming a bit "stiff." Nevertheless, it is quite satisfactory as it seems to be fairly consistent in performance quality. This is a desirable property, especially for higher-dimensional surfaces of which the results are hard to visualize.

4 Robust RBF Network Construction and Fine-Tuning

In this section, we discuss two ways to select only a portion of the training samples to be included as RBF kernels. One popular method [1, 2] is to include only a random subset of the available data as RBF kernels. For example, 70 of the 11 x11 samples are randomly selected as *seeding* kernels. To further even out the distribution of kernels, a *self-organizing* K-means clustering algorithm [1, 7] is used to establish K cluster centers and cluster sizes to cover all the 11 x 11 training samples. The initially seeded 70 kernels have now migrated to the 70 (K= 70) cluster centers with individually tailored cluster sizes σ's. This works quite well for the example considered here. Figure 5 compares the results with that obtained earlier using all the 11 x 11 kernels. Indeed, the 70-node K-means based result has learned a very nice and smooth response surface.

This K-means clustering based algorithm, however, has two potential problems. First, there is still an element of chance in getting the right kernels. Second, since clustering is more probability density oriented, this method was invented for pattern classification [1]. For function approximation, clustering does not necessarily guarantee good results, because two samples close to each other in the input space do not necessarily have similar outputs. To illustrate the second point, we consider a rather difficult response surface shown with 41 x 41 points in Figure 6. It is rather difficult to model that narrow peak well with any element of chance. To increase the chance of seeding the peak, we actually modified the random seeding process such that points with higher output values have higher chance of getting seeded. Using this biased seeding scheme, 30 samples are randomly picked as initial kernels. After K-means clustering and training with the complete set of 41 x 41 samples, the learned response surface is shown in Figure 6. Though qualitatively correct, it fails to capture the sharpness of the peak.

An *improved* backpropagation (BP) type of algorithm, based on recursive least squares (RLS) and optimal credit assignment [4], has been used to adjust the centers and σ's of the RBF kernels along with the output layer weights. After 30 BP iterations, a much better result is shown in Figure 6. Using BP on RBF network is usually much faster than on MP type network, because the hidden layer in a RBF network is already "trained" with the data. The added BP is therefore a viable tool for *fine-tuning* or adapting a RBF network.

A more systematic procedure is discussed below as the second method of selecting the RBF kernels. This recent procedure [5], known as the Orthogonal Least Squares (OLS), is a robust model building procedure similar to the ones used in building polynomial based regression models [6]. Using Gram-Schmidt type of orthogonal projection, it systematically selects the best kernels one at a time. The selection process terminates when the number of predetermined kernels have been filled or when the benefit of adding more kernels becomes diminishing. Figure 6 shows the result of us-

ing a *variant* OLS to select 30 kernels. The standard OLS in [5] uses ***unnormalized*** RBF and a single σ for all the kernels. We have allowed for indivial σ's and normalization. Certainly, comparing to the K-means trained result, the OLS result is much better and it is also better than the result obtained by adding 30 BP iterations. The price paid is somewhat more computation time (than K-means), depending on the number of training samples and implementation. However, if higher accuracy is desired, this ***initial*** investment in extra computation may be well worthwhile. Subsequent learning can be performed adaptively with efficiency on the output layer using recursive least squares (RLS), or on the hidden and the output layers with BP.

5 Connections and Extensions

In relation to the RBF networks, the concepts of potential functions [7] and Parzen windows [7] in nonparametric pattern recognition should be mentioned. These ideas have also been formulated as the generalized regression neural network (GRNN) [8]. The GRNN is closely related to the RBF network. It always uses normalization. However, it is based on a histogram accumulation type of concept, therefore does not allow the flexibility of training the output layer. Similar "sum of Gaussians" type of concept has also been used in nonlinear filtering theory.

A very important connection that deserves much attention is the relationship between RBF networks and ***fuzzy inference systems.*** Since each RBF kernel defines a ***localized receptive field*** in the input space, it forms the concept of a ***cluster*** or ***class.*** When stimulated by an input vector, the outputs of these RBF kernels are the ***class memberships*** of the input. Since the clusters or classes overlap, the class memberships are *fuzzy.* Each class membership, ϕ_i, would excite its respective output action, w_i. To obtain a unique output action, some form of ***defuzzification*** must be used. It turns out that RBF ***normalization*** and summation is a form of centroid defuzzification [9]. (The *multivariate* Gaussian can also be viewed as the *product* of single-variate Gaussians, thus performing *conjunctive* operations in the "premise" part of the fuzzy rules.) Therefore, the RBF network performs a form of fuzzy information processing.

This neuro-fuzzy connection benefits both neural network research and fuzzy system research. Neural Network benefits from it in (at least) the following ways. First, a fuzzy rule-based interpretation of the knowledge learned by a RBF network is now possible. Conversely, it is now possible to incorporate physical insight of the problem into a RBF network. That is, learning does not have to start from scratch. Second, the conjunctive-rule interpretation leads to the following extension. That is, in a RBF network, each RBF kernel can have a *vector* σ instead of a scalar one. This makes sense because the original RBF network implicitly assumes that all the input variable have been scaled to about the same range. Third, the work that has been done in the context of fuzzy logic control using ***piecewise linear*** actions [10] can now be readily transported to the study of (smoothed) ***piecewise-linear RBF networks.***

Fuzzy system research benefits from BP-type as well as OLS type of learning. Traditionally, fuzzy rule-based systems are often difficult to tune. Equipped with BP and OLS, one can ***automatically fine-tune a fuzzy system*** or construct one directly from examples.

References

[1] J.A. Leonard and M.A. Kramer, "Radial Basis Function Networks for Classifiying Process Faults," *IEEE Control Systems Magazine*, Vol. 11, No. 3, April 1991, pp. 31 - 38.

[2] J. Moody and C. Darken, "Learning with Localized Receptive Fields," *Proc. 1988 Connectionist Summer School (Carnegie-Mellon)*, Touretzky, Hinton and Sejnowski (Eds.), San Mateo, CA: Morgan Kaufmann, 1989, pp. 133 - 143.

[3] N. Dyn, D. Levin and S. Rippa, "Numerical Procedures for Surface Fitting of Scattered Data By Radial Functions," *SIAM J. Sci. Stat. Comput.*, Vol. 7, No. 2, April 1986, pp. 639 - 659.

[4] K.M. Tao and M. Jurik, "Improved Backpropagation Algorithms – a systems control and identification approach", *Record 23rd Asilomar Conf. Signals, Syst., Computers*, Nov. 1989, Pacific Grove, CA, pp. 111 - 115.

[5] S. Chen, C.F.N. Cowan, and P.M. Grant, "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks," *IEEE Trans. Neural Networks*, Vol. 2, No. 2, march 1991, pp. 302 - 309.

[6] M.S. Younger, *A Handbook for Linear Regression*, North Scituate, MA: Duxbury Press, 1979.

[7] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, NY: Wiley, 1973.

[8] D.F. Specht, "A General Regression Neural Network," *IEEE Trans. Neural Networks*, Vol. 2, No. 6, November 1992, pp. 568 - 576.

[9] L.-X. Wang and J.M. Mendel, "Fuzzy Basis Functions, Universal Approximations, and Orthogonal Least-Squares Learning," *IEEE Trans. Neural Networks*, Vol.3, No.5, Sept. 1992, pp. 807 - 814.

[10] J.-S.R. Jang, "Self-Learning Fuzzy Controllers Based on Temporal Back Propagation," *ibid*, pp. 714 - 723.

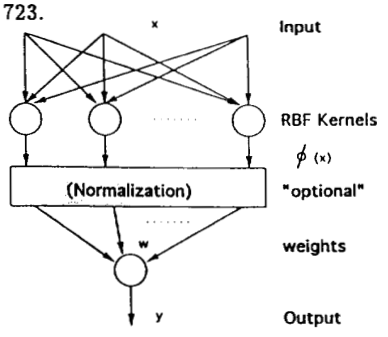

Figure 1: Radial Basis Function (RBF) Network

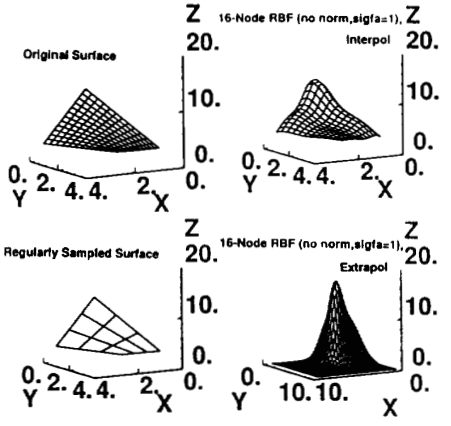

Figure 2: RBF Generalization (without Normalization)

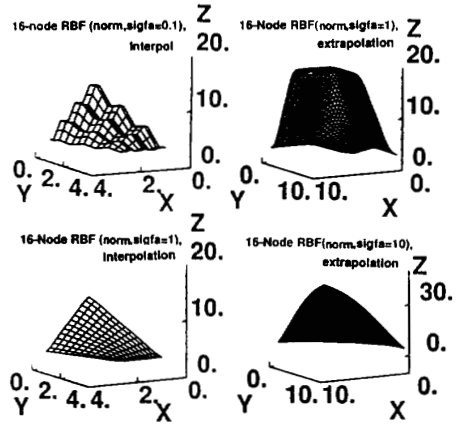

Figure 3: RBF Generalization (with Normalization)

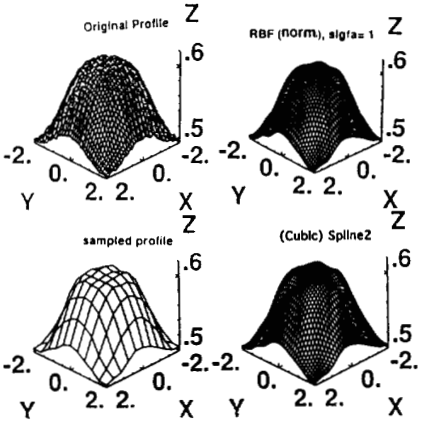

Figure 4: Response Surface Modelling (even distribution) -- Gaussian RBF vs (Cubic) Spline

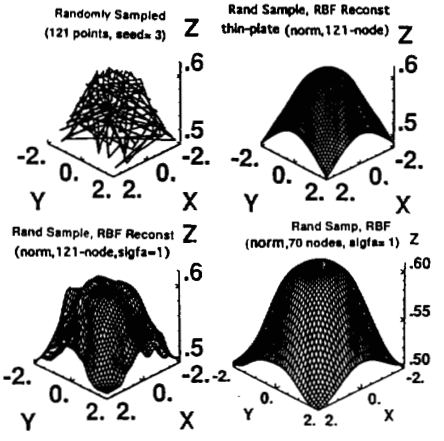

Figure 5: Response Surface Modelling (uneven distrib) -- Gaussian, thin-plate spline and subsampling

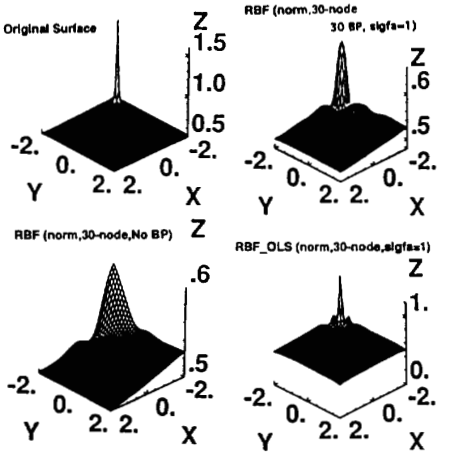

Figure 6: RBF Learning -- K-Means only, with BP, and OLS