BI-RADIAL TRANSFER FUNCTIONS

Włodzisław Duch & Norbert Jankowski
Department of Computer Methods
Nicholas Copernicus University

ul. Grudziądzka 5, 87–100 Toruń, Poland
phone: +48 56 21065 fax: +48 56 21543
e-mail: duch,norbert@phys.uni.torun.pl
www: http://www.phys.uni.torun.pl/~duch
http://www.phys.uni.torun.pl/~norbert

The most common transfer functions in neural networks are of the sigmoidal type. In this article other transfer functions are considered. Advantages of simple gaussians, giving hyperelliptical densities, and gaussian bar functions (sums of one-dimensional gaussians) are discussed. Bi-radial functions are formed from products of two sigmoids. Product of M bi-radial functions in N-dimensional parameter space give arbitrarily shaped densities offering great flexibility. Extensions of bi-radial functions are proposed. Bi-radial functions can be used as transfer functions in many neural networks, such as RBF, RAN or FSM systems.

1 Introduction

Adaptive systems of Artificial Neural Network (ANN) type were motivated by the parallel processing capabilities of the real brains, but the processing elements and the architectures used in artificial neural networks have little in common with biological structures. Artificial neural networks are networks of simple processing elements (usually called *neurons*) with internal adjustable parameters W. Modification of these adjustable parameters allows the network to learn an arbitrary vector mapping from the space of inputs X to the space of outputs $Y = A_W(X)$.

ANNs are adaptive systems with the power of a universal computer, i.e. they can realize an arbitrary mapping (association) of one vector space (inputs) to the other vector space (outputs). They differ in many respects, one of the most important characteristics being the transfer functions performed by each neuron. The first attempts at modeling of neural networks was via logical networks [12], i.e. using threshold devices performing step functions. These step functions were generalized in a natural way to functions of sigmoidal shape. Single-layer neural networks with sigmoidal functions are universal approximators [2, 10], i.e. they can approximate an arbitrary continuous function on a compact domain with arbitrary precision given sufficient number of neurons. The same result holds for the networks with neurons that give gaussian outputs instead of sigmoidal outputs [9, 14]. A new type of transfer functions, called *gaussian bars*, has been proposed by Hartman and Keeler [8]. None of these functions is flexible enough to describe an arbitrarily shaped density distributions of the multidimensional input space. The purpose of the activation and the transfer functions of neural elements is to enable the tessalation of the parameter space in the most flexible ways using the lowest number of parameters. The adaptive system is a collection of communicating elements and

the processing function of a single element is the most important characteristic of the whole system.

In this paper we investigate various other simple functions suitable as the transfer functions of neurons. In the next section the non-local transfer functions used in literature are reviewed and some new possibilities discussed. In the third section description of local and semi-local processing units functions is presented and bi-radial functions and their extensions are introduced. The fourth section presents results obtained using different transfer functions in the RBF-type of networks.

2 Non-local Transfer Functions

Two functions determine the way signals are processed by neurons. *The activation function* determines the total signal neuron receives. In most cases a fan-in function, i.e. a linear combination of the incoming signals, is used. For neuron i connected to neurons j (for $j = 1, \ldots, N$) sending signals x_j with the strength of the connections w_{ij} the total activation signal I_i is

$$I_i(\mathbf{x}) = \sum_{j=1}^{N} w_{ij}\, x_j \tag{1}$$

The second function determining neuron's signal processing is *the output function* $o(I)$. These two functions together determine the values of the neuron outgoing signals. The total function acts in the N-dimensional *input space*, called also *the parameter space*. The composition of these two functions is called the *transfer function* $o(I(\mathbf{x}))$. The activation and the output functions of the input and the output layers may be of different type than those of the hidden layer, in particular frequently linear functions are used for inputs and outputs and non-linear output functions for hidden layers.

The first neural network models proposed in the 40-ties by McCulloch and Pitts [12] were based on the logical processing elements of the threshold type. The output function of the logical elements is of *the step function* type, and is known also as the Heaviside $\Theta(x)$ function: it is 0 below the threshold value and 1 above it. The use of such functions was motivated by the logical analysis of the computing circuits and the methaphore (very popular in the early days of computers) of brains seen as computers. In principle one can perform arbitrary computations using logical neurons. Real values may be quantized and the logical neurons used to learn the bits. The greatest advantage of the logical elements is the speed of computations and the possibility to realize relatively easy some functions in hardware. Classification regions of the logical networks are of the hyperplane type rotated by the w_{ij} coefficients.

An intermediate multi-step type of functions between continuous sigmoidal functions and step functions are sometimes used, with a number of thresholds. Instead of the step function semi-linear functions were used and later generalized to *the sigmoidal functions*, leading to the *graded response neurons*:

$$\sigma(x; s) = \frac{1}{1 + e^{-x/s}} \tag{2}$$

The constant s determines the slope of the sigmoidal function around the linear part. This function may also be replaced by the arcus tangent or the hyperbolic tangent function:

$$\tanh(x;s) = \frac{1-e^{-x/s}}{1+e^{-x/s}} \tag{3}$$

Other sigmoidal functions may be useful to speed up computations:

$$s_1(x;s) = \Theta(x)\frac{x}{x+s} - \Theta(-x)\frac{x}{x-s} = x\frac{\mathrm{sgn}(x)x-s}{x^2-s^2} \tag{4}$$

$$s_2(x;s) = \frac{\sqrt{1+s^2x^2}-1}{sx} \tag{5}$$

where $\Theta(x)$ is a step function.

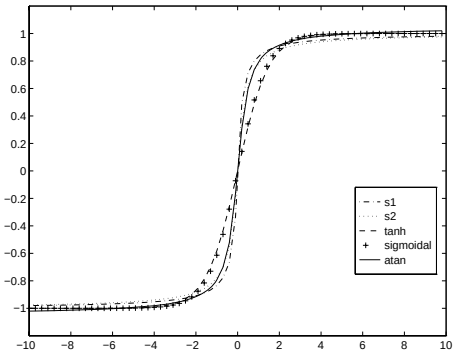

Figure 1: Comparison of non-local transfer functions.

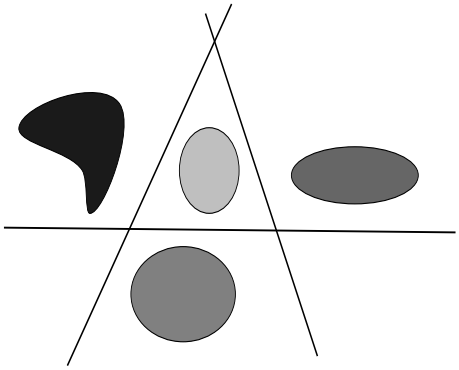

Figure 2: Decision regions formed using sigmoidal processing functions.

Shapes of these functions[1] are compared in Fig. 1. The sigmoidal function and the hyperbolic tangent functions are hard to distinguish in this figure while the arcus tangent and the s_1, s_2 functions change asymptotically reaching saturation for larger activation values more slowly. All these functions are very similar and all what we recommend the use of s_1 or s_2 functions since their computational costs are lower.

Sigmoidal functions have non-local behavior, i.e. they are non-zero in infinite domain. The decision regions for classification are formed by cutting the parameter space with the hyperplanes (Fig. 2). The system *pretends* that it knows everything, which is quite false especially far from the sample data regions where hyperplanes, extending to infinity, enforce arbitrary classifications. Sigmoidal output functions smooth out many shallow local minima in the total output functions of the network. For classification problems this is very desirable, but for general mappings it limits the precision of the adaptive system.

For sigmoidal functions powerful mathematical results exist showing that a universal approximator may be built from only single layer of processing elements [2, 10]. Another

[1] All these functions are linearly transformed to obtain output between -1 and 1 and different slope parameters s are used to show that all functions are quite similar.

class of powerful functions used in approximation theory [16, 5, 6] is called the radial basis functions (RBFs). Some of these functions are non-local while most are localized. RBF networks are also universal approximators [9, 14]. Admitting processing units of the sigma-pi type higher-order products of inputs are taken into account and the approximating function becomes a product of various powers of input signals [4].

3 Local and Semi-local Transfer Functions

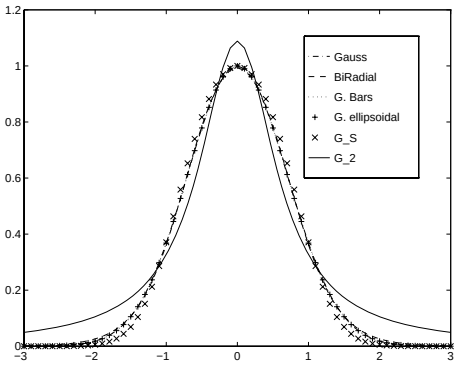

Figure 3: Comparison of several localized functions fitted to a gaussian.

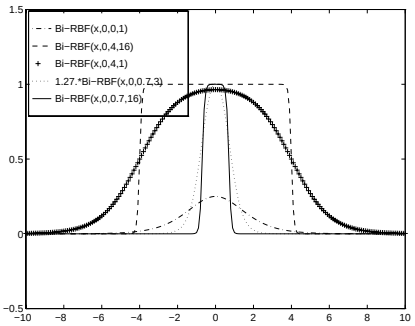

Figure 4: A few shapes of the bi-radial functions in one dimension.

From the point of view of an adaptive system used as a classification device one can either divide the total parameter space into regions of classification using non-local functions or set up local regions around the data points. Each of these approaches has some advantages and disadvantages. A few attempts were made to use localized functions in the adaptive systems, some of them may be traced back to the older work on pattern recognition [7]. Moody and Darken [13] used locally-tuned processing units to learn real-valued mappings and classifications in a learning method combining self-organization and supervised learning. They have selected locally-tuned units to speed up the learning process of backpropagation networks. Botttou and Vappnik [1] showed the power of local training algorithms in a more general way.

Although the processing power of neural networks based on non-local processing units does not depend strongly on the type of neuron processing functions such is not the case for localized units. Gaussian functions are perhaps the simplest but not the least expensive to compute. Simple quadratic and quartic functions approximate roughly the shape of gaussian function:

$$g_2(\mathbf{x},\mathbf{t},s) = \frac{1}{1+||\mathbf{x}-\mathbf{t}||^2/s^2}; \quad g_4(\mathbf{x},\mathbf{t},s) = \frac{1}{1+||\mathbf{x}-\mathbf{t}||^4/s^2} \tag{6}$$

3.1 Radial Basis Functions (RBFs)

Radial Basis Functions are used as transfer functions in many neural network simulators. These types of functions have been in use in approximation theory [16, 5, 6] and in pattern recognition under different names for many years (cf. potential function approach, [7]). A very good introduction to RBF and more general regularization networks was given by Poggio and Girosi [15]. A few types of localized radial basis functions exist, among them the *gaussian functions* (Eq. 10). Examples of the Radial Basis Functions include the nonlocal radial coordinates, multiquadratics, gaussians and thin-plate spline functions:

$$
\begin{aligned}
h_1(\mathbf{x};\mathbf{t}) &= ||\mathbf{x}-\mathbf{t}|| && (7)\\
h_2(\mathbf{x};\mathbf{t},b) &= (b^2+||\mathbf{x}-\mathbf{t}||^2)^{-\alpha}, \quad \alpha>0 && (8)\\
h_3(\mathbf{x};\mathbf{t},b) &= (b^2+||\mathbf{x}-\mathbf{t}||^2)^{\beta}, \quad 0<\beta<1 && (9)\\
h_4(\mathbf{x};\mathbf{t},b) &= e^{-||\mathbf{x}-\mathbf{t}||^2/b^2} && (10)\\
h_5(\mathbf{x};\mathbf{t},b) &= (b||\mathbf{x}-\mathbf{t}||)^2\ln(b||\mathbf{x}-\mathbf{t}||) && (11)
\end{aligned}
$$

The simplest approach, used in the RBF networks, is to set a number of radial functions $G_i(\mathbf{x})$ with predetermined dispersions and positions (for example, positions are set by k-means clustering and dispersions to twice the nearest neighbor distance) and determine the linear coefficients w_i in the approximation function

$$f(\mathbf{x};\mathbf{w},\mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x},\mathbf{p}_i). \tag{12}$$

In the regularization networks also the centers of each of the radial units are optimized [15], allowing for reduction of the number of centers in the presence of noisy data (corresponding to the regularization of approximating function). Thus in N-dimensional case a center is described by N coordinates and one dispersion. A straightforward generalization of the radial units of the gaussian type is to allow output functions with different dispersions for different dimensions, giving $2N$ adaptive parameters, centers and dispersions, per one unit.

3.2 Ellipsoidal activation functions

Changing the activation function $I(x_i)$ to the quadratic activation:

$$I(\mathbf{x};\mathbf{t},\mathbf{w}) = \sum_i w_i(x_i-t_i)^2 \tag{13}$$

allows to use the multivariate gaussian function to obtain ellipsoidal output densities:

$$G_g(\mathbf{x};\mathbf{t},\mathbf{b}) = e^{-I(\mathbf{x};\mathbf{t},\mathbf{w})} = \prod_{i=1}^{N} e^{-(x_i-t_i)^2/b_i^2} \tag{14}$$

with $w_i = 1/b_i^2$. Similar result is obtained by combining the sigmoidal output function with quadratic activation:

$$G_S(\mathbf{x};\mathbf{t},\mathbf{b}) = 2\left(1-\sigma(I(\mathbf{x};\mathbf{t},\mathbf{w}))\right) = 2 - \frac{2}{1+e^{-\sum_{i=1}^{N}(x_i-t_i)^2/b_i^2}} \tag{15}$$

Simpler units giving ellipsoidal densities are also possible, for example:

$$G_2(\mathbf{x};\mathbf{t},\mathbf{b}) = \prod_{i=1}^{N} \frac{1}{1+(x_i-t_i)^2/b_i^2} \tag{16}$$

A number of local training algorithms may be devised for such transfer functions combining the k-means clustering for initial placements of ellipsoids in a self-organizing fashion, followed by growing and pruning of the new ellipsoidal units in supervised algorithm. In particular if the training algorithm localizes neuron processing function in the region far from the given data points the unit may be removed without loss. For N-dimensional input space each ellipsoidal unit uses $2N$ adaptive parameters.

3.3 Bar functions

The problem of noisy dimensions in RBF networks, i.e. irrelevant inputs that do not contribute to the determination of the output values, has been addressed by Hartman and Keeler [9] and by Park and Sandberg [14]. Instead of multidimensional gaussian functions these authors advocate a combination of one-dimensional gaussians:

$$G_b(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{w}) = \sum_{i=1}^{N} w_i e^{-(x_i-t_i)^2/b_i^2} \tag{17}$$

The activation and the output functions are inseparable in this case. $3N$ adjustable parameters are needed per processing unit. These functions are called *gaussian bar functions* because except for a single maximum around center $\mathbf{t}$ in N-dimensions they include gaussians in $N-1$ dimensional subspace. For large number of dimensions N these bars have values w_i that may be much lower than the sum of all weights w_i. The network output may be processed via sigmoidal function removing these bars.

Gaussian bars make elimination of irrelevant input variables, i.e. dimensionality reduction, easier than in the multidimensional gaussian case, although variable dispersions should allow to reduce some of the dimensions to zero (cf. the example of quadratic logistic mapping given by Moody and Darken [13]). Another advantage of using the bar functions follows from the very existence of these bars. A single maximum or a few separated maxima are described by a small number of gaussian functions with only $N+1$ parameters each and require the same number of gaussian bar functions with almost three times as many parameters. However, if there are k regularly spaced input clusters in each dimension in the N-dimensional hypercube kN clusters are formed, and each should be represented by a separate multivariate gaussian. On the other hand kN gaussian bar functions are sufficient to describe such a case.

Similar combination of sigmoidal functions will create a *sigmoidal bar function*. Such functions should not be used to represent data clustered around a few points only because each cluster requires $2N$ sigmoidal functions while one gaussian function may be sufficient to model a cluster. However, if the data clusters are regularly spaced in a quadratic mesh,

with k^2 clusters each will need a separate gaussian while $2 \cdot 2k = 4k$ sigmoidal *bars* in the input space are sufficient to represent such data.

3.4 Bi-radial functions

Rather than a single sigmoidal function one may use products of pairs of sigmoidal functions for each variable. This type of output functions is the most flexible, producing decision regions of arbitrary shapes for classification. Product of $2N$ sigmoids has the following general form:

$$Bi(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) = \prod_{i=1}^{N} \sigma(e^{s_i} \cdot (x_i - t_i + e^{b_i}))(1 - \sigma(e^{s_i} \cdot (x_i - t_i - e^{b_i}))) \qquad (18)$$

where $\sigma(x) = 1/(1+e^{-x})$. The first sigmoidal factor in the product is growing for increasing input x_i while the second is decreasing, localizing the function around t_i. Shape adaptation of the density $Bi(\mathbf{x},\mathbf{t},\mathbf{b},\mathbf{s})$ is possible by shifting centers $\mathbf{t}$, rescaling $\mathbf{b}$ and $\mathbf{s}$. The number of adjustable parameters per processing unit is in this case (not counting the weights w_i) $3N$. Dimensionality reduction is possible as in the *gaussian bar case*, but we can obtain more flexible density shapes, thus reducing the number of adaptive units in the network. Exponentials e^{s_i} and e^{b_i} are used instead of s_i and b_i to prevent oscillations during learning procedure (learning becomes more stable).

It is possible to extend the localized bi-radial functions to the semi-localized functions:

$$S - Bi((\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) = \prod_{i=1}^{N} (\alpha + \sigma(e^{s_i} * (x_i - t_i + e^{b_i})))(1 - \beta\sigma(e^{s_i} * (x_i - t_i - e^{b_i}))). \qquad (19)$$

This function does not vanish for large $|x|$, for $\alpha = 0$, $\beta = 1$ is identical to the bi-radial localized functions while for $\alpha = \beta = 0$ it turns into sigmoidal function. At the beginning of learning procedure α and β are equal to zero). Semi-local function $S - Bi$ have $5N$ parameters for each units.

4 RBF network with bi-radial functions

In figures 5 and 6 convergence of errors during learning obtained with gaussian(Eq. 10, one common dispersion per function), sigmoidal (Eq. 15) (separate dispersion per function) and bi-radial functions (Eq. 18) are presented. The same RBF-type of network was used, with each case trained for 2000 epochs on the two-spiral classifcation benchmark. This is a difficult test for backpropagation networks. the number of classifcation points is 196, points are divided in two classes and the number of network nodes is set two 100. The network based on bi-radial transfer functions not only learns faster (Fig 5) but also generalizes better (Fig 6).

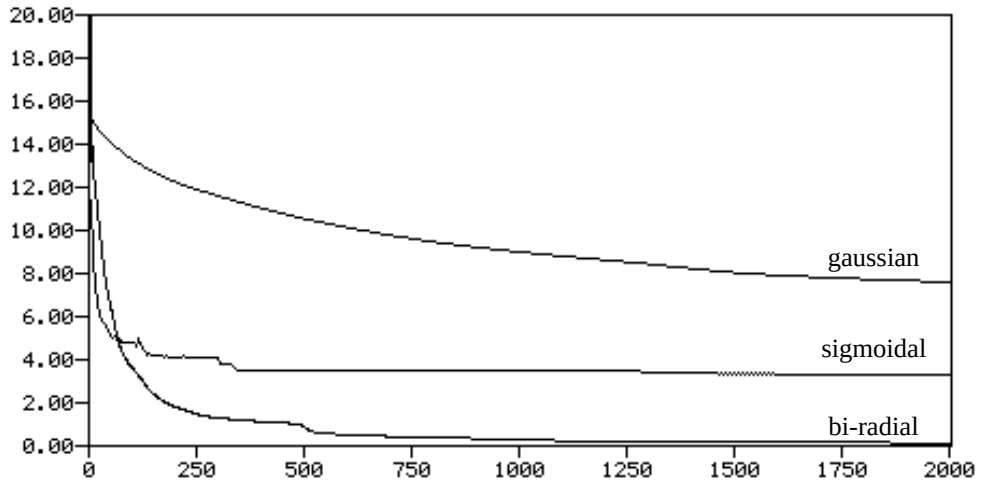

Figure 5: Comparison of the summed squared errors for different transfer functions: gaussian (Eq. 10), sigmoidal (Eq. 15) and bi-radial function (Eq. 18) used in the same RBF net during 2000 epochs.

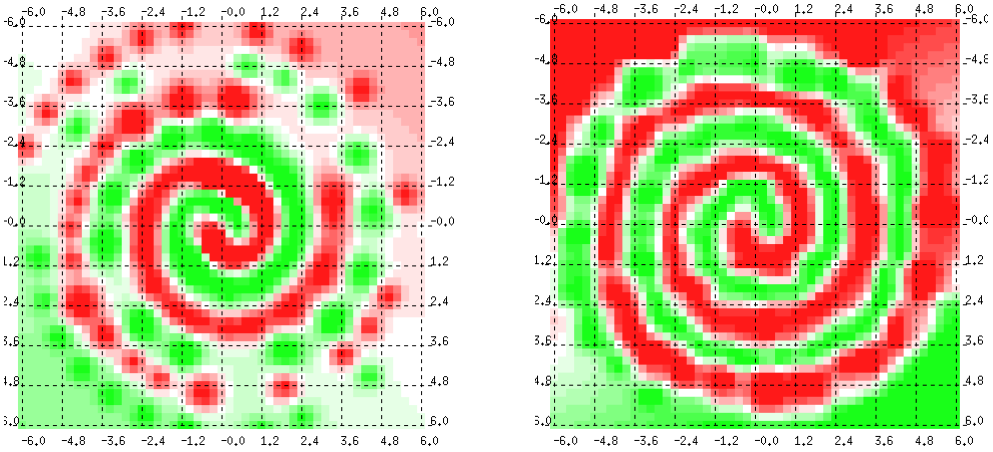

Figure 6: Results for the two spiral classification problem solved with gaussian (on the left) and bi-radial (on the right) transfer functions.

5 Summary

We have presented several transfer functions suitable for neural units. Localized neuron output functions seem to be quite efficient in describing arbitrary decision regions of neural nets used for mapping or classification purposes. New type of transfer functions proposed here – the bi-radial functions – contain $3N$ parameters per one unit and are quite flexible, representing various densities of the input data. Semi-bi-radial functions provide local and non-local units in one network. Next step towards greater flexibility requires rotation of each

unit separately or even a general form of the quadratic activation function:

$$I(\mathbf{x};\mathbf{t},\mathbf{w}) = \sum_{ij} w_{ij}(x_i - t_i)(x_j - t_j) \tag{20}$$

For rotation this adds at least $N-1$ parameters for a total of $3N-1$ parameters, while for general rotation and arbitrary rescaling N^2 parameters are added. So far we have not seen any adaptive systems using such generalized output functions. There is a tradeoff between the flexibility of the processing units connected with the number of adjustable parameters and the complexity of the learning process of the whole network.

In the near future we shall use bi-radial and semi-bi-radial transfer functions in the FSM [3] and IncNet architecture (RBF net with statistically controlled growth of units number) [11] and extend IncNet by including pruning of some nodes.

References

[1] BOTTOU, L., AND VAPNIK, V. Local learning algorithms. *Neural Computation 4*, 6 (1992), 888–900.

[2] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems 2*, 4 (1989), 303–314.

[3] DUCH, W., AND DIERCKSEN, G. H. F. Feature space mapping as a universal adaptive system. *Computer Physics Communications 87* (5 1994), 341–371.

[4] DURBIN, R., AND RUMELHART, D. E. Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation 1* (1989), 133–142.

[5] DYN, N. Interpolation and approximation by radial and related functions. In *Approximation Theory VI*, C. K. Chiu, L. L. Schumaker, and J. D. Watts, Eds. Academic Press, 1989.

[6] FRANKE, R. Scattered data interpolation: test of some methods. *Math Computation 38* (1982), 181–200.

[7] FUKUNAGA, K. *Introduction to Statistical Pattern Recognition*. Academic Press, 1972.

[8] HARTMAN, E., AND KEELER, J. D. Predicting the future: Advantages of semilocal units. *Neural Computation 3*, 4 (1991), 566–578.

[9] HARTMAN, E. J., KEELER, J. D., AND KOWALSKI, J. M. Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation 2*, 2 (1990), 210–215.

[10] HORNIK, K., STINCHCOMBE, M., AND WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 5 (1989), 359–366.

[11] KADIRKAMANATHAN, V. A statistical inference based growth criterion for the RBF network. In *Proc. IEEE. Workshop on Neural Networks for Signal Processing* (1994).

[12] MCCULLOCH, W. S., AND PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5* (1943), 115–133.

[13] MOODY, J., AND DARKEN, C. J. Fast learning in networks of locally-tuned processing units. *Neural Computation* (1989), 281–294.

[14] PARK, J., AND SANDBERG, I. W. Universal approximation using radial-basis-function networks. *Neural Computation 3*, 2 (1991), 246–257.

[15] POGGIO, T., AND GIROSI, F. Network for approximation and learning. *Proc. IEEE 78*, 9 (Sept. 1990), 1481–1497.

[16] POWELL, M. J. D. Radial basis functions for multivariable interpolation: A review. In *Algorithms for Approximation of Functions and Data* (1987), M. J. C and C. M. G., Eds., Oxford University Press, pp. 143–167.