NEW NEURAL TRANSFER FUNCTIONS

Włodzisław Duch & Norbert Jankowski

Department of Computer Methods, Nicholas Copernicus University,
ul. Grudziądzka 5, 87-100 Toruń, Poland
e-mail: duch,norbert@phys.uni.torun.pl, http://www.phys.uni.torun.pl/kmk

The choice of transfer functions in neural networks is of crucial importance to their performance. Although sigmoidal transfer functions are the most common there is no *a priori* reason why they should be optimal in all cases. In this article advantages of various neural transfer functions are discussed and several new type of functions are introduced. Universal transfer functions, parametrized to change from localized to delocalized type, are of greatest interest. Biradial functions are formed from products or linear combinations of two sigmoids. Products of N biradial functions in N-dimensional input space give densities of arbitrary shapes, offering great flexibility in modelling the probability density of the input vectors. Extensions of biradial functions, offering good tradeoff between complexity of transfer functions and flexibility of the densities they are able to represent, are proposed. Biradial functions can be used as transfer functions in many types of neural networks, such as RBF, RAN, FSM and IncNet. Using such functions and going into the hard limit (steep slopes) facilitates logical interpretation of the network performance, i.e. extraction of logical rules from the training data.

Keywords: neural networks, adaptive systems, local learning, transfer functions, RBF nets, IncNet, FSM, extraction of logical rules.

1 Introduction

Adaptive systems of the Artificial Neural Network (ANN) [15] type were initially 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. ANNs 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 $Y = A_W(X)$. From a probabilistic point of view adaptive systems should approximate the density of joint probability $p(X, Y)$ or the posterior probability $p(Y|X)$. Flexible estimation of densities is thus of primary importance.

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 made using logical networks [25], or 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 [5, 16], 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 [14, 29]. A new type of transfer functions, called *gaussian bars*, has been proposed by Hartman and Keeler [13]. In the *functional link* networks of Pao [28] a combination of various functions, such as polynomial, periodic, sigmoidal and Gaussian functions are used. *Rational transfer functions* were used by Haykin and Leung with very good results [21]. In the *conic section* function networks Dorffner [6] introduced functions that change smoothly from sigmoidal to Gaussian-like. *Lorentzian transfer functions*, which may be treated as a simplified Gaussian functions, were used by Giraud et al. [12]. Nonmonotonic transfer function have been recently used by Morita [27].

There is a growing understanding that the choice of transfer functions is at least as important as the network architecture and learning algorithm. Neural networks are used either to approximate *a posteriori* probabilities for classification or to approximate probability densities of the training data [34]. None of the functions mentioned above is flexible enough to describe an arbitrarily shaped density distributions of the multidimensional input space. Viewing the problem of learning from geometrical point of view the purpose of the transfer functions performed by the neural network nodes is to enable the tessellation of the parameter space in the most flexible way using the lowest number of adaptive parameters. Implications of this fact has not yet been fully understood by many researchers.

In this paper we investigate various functions suitable as the transfer functions for neural networks. Systematic investigation of transfer functions is a fruitful task. Since information about various transfer functions is scattered in the literature and has not been reviewed so far we have collected and commented upon a number of transfer functions alternative to sigmoidal functions. To keep the paper rather short nonmonotonic transfer functions have been omitted here, although they may actually be more faithful to neurobiology and may help to avoid the local minima of the neural network error function ([27], Duch and Ludwiczewski, in preparation). In the next section a few non-local transfer functions are described and their simplified versions discussed. In the third section description of local and semi-local processing functions is presented and biradial transfer functions are introduced. The fourth section presents results obtained using different transfer functions in several RBF-type of networks. Short discussion concludes this paper.

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 this section 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 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 neuron processing 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 layers, in particular frequently linear functions are used for inputs and outputs and non-linear transfer functions for hidden layers.

The first neural network models proposed in the forties by McCulloch and Pitts [25] were based on the logical processing elements. The output function of the logical elements is of *the step function* type, and is known also as the Heaviside $\Theta(I-\theta)$ function: it is 0 below the threshold value θ and 1 above it. The use of such threshold functions was motivated by the logical analysis of computing circuits and the metaphor (very popular in the early days of computers) of brains seen as networks of logical switching elements. In principle one can perform arbitrary computations using logical neurons. Real values should be quantized and the logical neurons used to learn the bits. The greatest advantage of using logical elements is the high speed of computations and the possibility to realize relatively easily some functions in the hardware. Classification regions of the logical networks are of the hyperplane type rotated by the W_{ij} coefficients.

Multi-step functions are an intermediate type of functions, between the step functions and semi-linear functions. Multi-step functions have a number of thresholds, $\varsigma(I) = y_i$ if $\theta_i \leq I < \theta_{i+1}$. To avoid evaluation of the logical IF conditions for constant difference $\theta = \theta_i - \theta_{i+1}$ multi-step functions are efficiently implemented using auxiliary step vectors $\mathbf{v}$ and integer arithmetics to convert rescaled input values to arbitrary output values: $\mathbf{v}\,[\Theta\,(1 + Int[(I-\theta_1)/\theta])]$, where θ_1 is the first threshold.

Instead of the step functions semi-linear functions are also used, $s_l(I;\theta_1,\theta_2) = \{0 \text{ for } I \leq \theta_1,\ (I-\theta_1)/(\theta_2-\theta_1) \text{ for } \theta_1 < I \leq \theta_2 \text{ and } 1 \text{ for } I > \theta_2\}$. These functions were later generalized to *the sigmoidal functions*, leading to the *graded response neurons*, used most often in the literature:

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

The constant s determines the slope of the sigmoidal function around the linear part. It is commonly believed that the activity of biological neurons follows such sigmoidal transfer function, but this is not the reason why sigmoidal functions became so popular. These functions are smooth and it is easy to calculate their derivatives, equal to $\sigma(I)' = \sigma(I)(1-\sigma(I))$. Sigmoidal functions may also be replaced by the arcus tangent or the hyperbolic tangent functions:

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

$$\tanh'(I/s) = \text{sech}^2(I/s)/s = \frac{4}{s(e^{-I/s}+e^{+I/s})^2} \tag{4}$$

Since calculation of exponents is much slower than simple arithmetic operations other functions of sigmoidal shape may be useful to speed up computations:

$$s_1(I;s) = \Theta(I)\frac{I}{I+s} - \Theta(-I)\frac{I}{I-s} = I\frac{\text{sgn}(I)I - s}{I^2-s^2} \tag{5}$$

$$s_2(I;s) = \frac{sI}{1+\sqrt{1+s^2I^2}} = \frac{sI}{1+q} \tag{6}$$

where $\Theta(I)$ is a step function and $q = \sqrt{1+s^2I^2}$. The derivative of these functions are also easy to compute:

$$s_1'(I;s) = \frac{s}{(I+s)^2}\Theta(I) + \frac{s}{(I-s)^2}\Theta(-I) = \frac{s}{(I+\mathrm{sgn}(I)s)^2} \quad (7)$$

$$s_2'(I;s) = \frac{s}{q(1+q)} \quad (8)$$

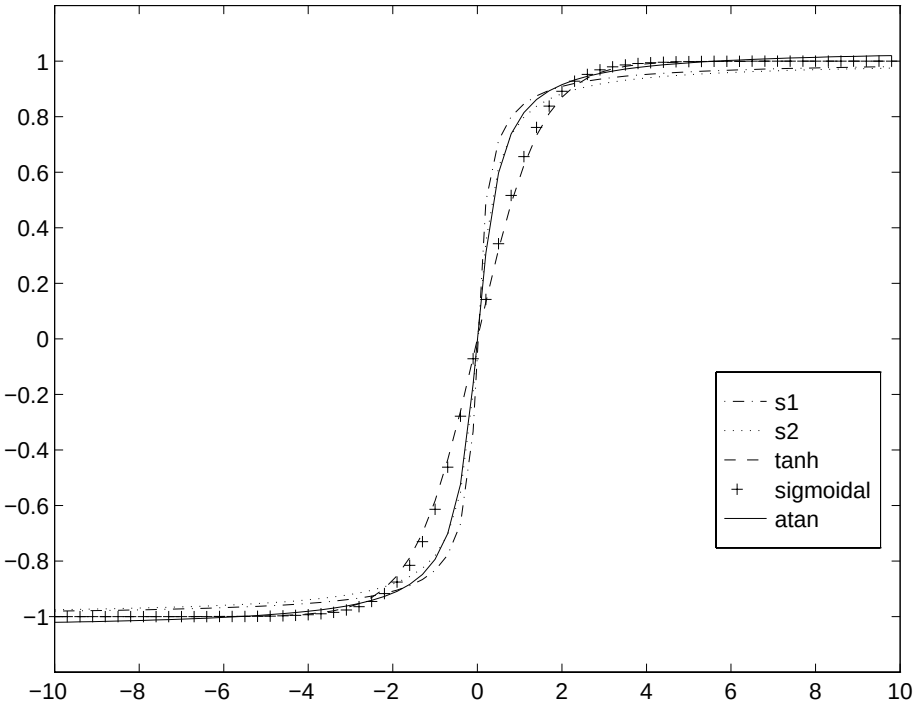

Figure 1: Comparison of non-local transfer functions.

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 therefore one may recommend the use of s_1 or s_2 functions since their computational costs are the lowest – in practical computations avoiding calculation of exponential factors one can gain a factor of 2-3.

Sigmoidal functions have non-local behavior, i.e. they are non-zero in infinite domain. The classification decision regions of neural networks based on these functions are formed by cutting the input space with hyperplanes (Fig. 2). The system *pretends* that it knows everything – this may be quite improper 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

[1] All these functions are linearly transformed to obtain output between -1 and 1; their slope parameters s are chosen to make them as similar to each other as possible.

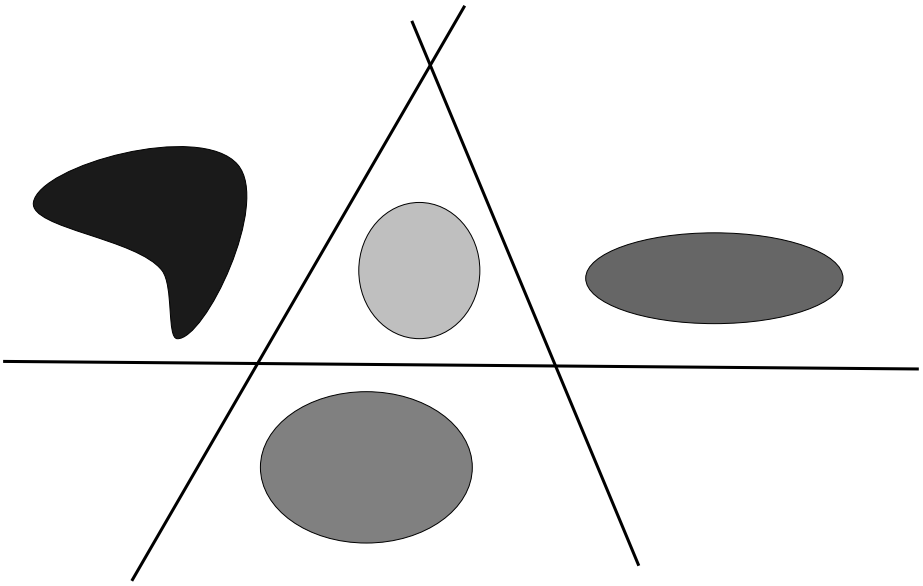

Figure 2: Decision regions formed using sigmoidal processing functions.

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 [5, 16]. Another class of powerful functions used in approximation theory [32, 9, 10] is called the radial basis functions (RBFs). Some of these functions are non-local while most are localized. RBF networks are also universal approximators [14, 29]. 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 [8].

For approximation problems Allison [1] recommends simple multiquadratic functions, similar to the $s_2(I;s)$ function:

$$s_m(I;\Delta) = \sqrt{I^2+\Delta^2}; \qquad s'_m(I;\Delta) = \frac{I}{s_m(I;\Delta)} \tag{9}$$

where Δ is the smoothness parameter.

3 Local and Semi-local Transfer Functions

Non-local transfer functions used in neural networks divide the total input space into regions corresponding to different classes or values of the output vector. A single adaptive parameter may change the output of the network at all points of the input space. Therefore the learning process must always change all adaptive parameters in a correlated way. Such transfer functions are used in multi-layered perceptrons (MLPs) for discrimination and approximation. Localized transfer functions use adaptive parameters that have only local influence on the network output, i.e. the output is changed only in the localized regions of the input space. Such functions are used in Gaussian-based radial basis function (RBF) networks, where classification and approximation is based

on prototypes rather than discriminant functions. In such networks the activation function is usually changed from the fan-in function to a distance function:

$$D_i(\mathbf{x}) = d(\mathbf{x}, \mathbf{t}_i) = ||\mathbf{x} - \mathbf{t}_i|| \tag{10}$$

where $\mathbf{t}_i$ is the center of the i-th unit, an adaptive parameter around which activation has large values. In practice Euclidean distance is used most often for the real-valued inputs and Hamming distance is frequently used for binary inputs. Additional adaptive parameters may be introduced as scaling factors in each dimension (N parameters), or as one common scaling factor for each center. For Euclidean distance $2N$ adaptive parameters are defined:

$$D^2(\mathbf{x}; \mathbf{t}, \mathbf{V}) = \sum_i V_i (x_i - t_i)^2 \tag{11}$$

A few attempts were made to use localized functions in adaptive systems. Some of them may be traced back to the older work on pattern recognition [11]. Moody and Darken [26] 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. Bottou and Vapnik [3] have shown the power of local training algorithms in a more general way. According to Kadirkamanathan and Niranjan [18] smoothness conditions for adding new units in constructive neural networks are satisfied only by strongly local units.

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 $e^{-D(x)^2}$ are perhaps the simplest but not the least expensive to compute. Simple quadratic and quartic functions approximate roughly the shape of a Gaussian function:

$$G_2(D(\mathbf{x})) = \frac{1}{1 + D^2(\mathbf{x})}; \quad G_2'(D) = -2DG_2^2(D); \tag{12}$$

$$G_4(D(\mathbf{x})) = \frac{1}{1 + D^4(\mathbf{x})}; \quad G_4'(D) = -4D^3G_4^2(D) \tag{13}$$

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 [32, 9, 10] and in pattern recognition under different names for many years (cf. potential function approach, [11]). A very good introduction to RBF and more general regularization networks was given by Poggio and Girosi [31] (see also [15, 4, 6, 22, 23, 24, 2, 30]). Several types of localized radial basis functions exist. They all treat the activation value as radial coordinate $r = ||\mathbf{x} - \mathbf{t}_i||$. Among them *Gaussian functions* (Eq. 14) are unique since for Euclidean distance functions (and other distance functions that may be presented as a sum of independent components) they are separable. Other examples of the Radial Basis Functions include the nonlocal radial coordinates, general multiquadratics, and thin-plate spline functions:

$$h_1(\mathbf{x}; \mathbf{t}, b) = e^{-||\mathbf{x}-\mathbf{t}||^2/b^2} \tag{14}$$

$$h_2(\mathbf{x};\mathbf{t}) = ||\mathbf{x}-\mathbf{t}|| \quad (15)$$

$$h_3(\mathbf{x};\mathbf{t},b) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{-\alpha}, \quad \alpha>0 \quad (16)$$

$$h_4(\mathbf{x};\mathbf{t},b) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{\beta}, \quad 0<\beta<1 \quad (17)$$

$$h_5(\mathbf{x};\mathbf{t},b) = (b||\mathbf{x}-\mathbf{t}||)^2 \ln(b||\mathbf{x}-\mathbf{t}||) \quad (18)$$

The simplest approach, used in the RBF networks, is to set a number of radial functions $G_i(\mathbf{x})$ with predetermined parameters b and positions $\mathbf{t}$ (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:

$$f(\mathbf{x};\mathbf{W},\mathbf{b},\mathbf{t}) = \sum_{i=1}^{M} W_i G_i(\mathbf{x},\mathbf{b}_i,\mathbf{t}) = \sum_{i=1}^{M} W_i e^{-||\mathbf{x}-\mathbf{t}_i||^2/b_i^2}. \quad (19)$$

In the regularization networks also the centers $\mathbf{t}_i$ of each of the radial units are optimized [31], allowing for reduction of the number of basis functions in the presence of noisy data (corresponding to the regularization of approximating function). Thus in the N-dimensional case a center is described by N coordinates $\mathbf{t}_i$ and one parameter b_i (dispersion for Gaussians). A straightforward generalization of the radial units of the Gaussian type with Euclidean distance function is to allow different dispersions for different dimensions, giving $2N$ adaptive parameters, or centers and dispersions, per each neural unit.

3.2 Gaussian and sigmoidal 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 [14] and by Park and Sandberg [29]. Instead of multidimensional Gaussian functions these authors advocate a combination of one-dimensional Gaussians:

$$G_b(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{v}) = \sum_{i=1}^{N} v_i e^{-(x_i-t_i)^2/b_i^2} \quad (20)$$

In this case the activation and the output functions are inseparable. $3N$ 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 subspaces. For large number of dimensions N these bars have values v_i that may be much lower than the sum of N weights v_i around $\mathbf{t}$. To smooth the network output and remove small maxima in the output layer sigmoidal function are used.

Gaussian bars make elimination of irrelevant input variables, i.e. dimensionality reduction, easier than in the multidimensional Gaussian case. Variable dispersions should also allow to reduce some of the dimensions to zero (cf. the example of quadratic logistic mapping given by Moody and Darken [26]). 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.

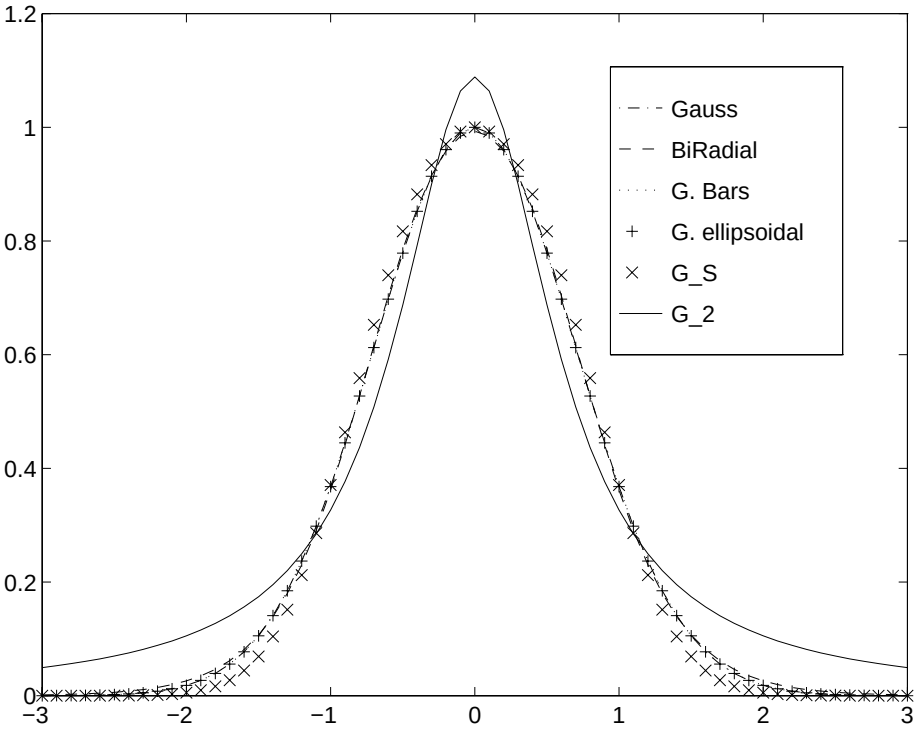

Figure 3: Comparison of several localized functions fitted to a gaussian.

Similar combination of sigmoidal functions will create *sigmoidal bar function*:

$$\sigma_b(\mathbf{x};\mathbf{t},\mathbf{W},\mathbf{v}) = \sum_{i=1}^{N} \frac{v_i}{1+e^{-W_i(x_i-t_i)^2/b_i^2}} \tag{21}$$

These functions, similarly as Gaussian bars, give surfaces of constant densities that cannot easily be rotated, which is clearly a disadvantage. Sigmoidal bar 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 each of the k^2 clusters should be represented by a separate Gaussian, while $2 \cdot 2k = 4k$ sigmoidal *bars* in the input space are sufficient to represent such data.

3.3 Ellipsoidal density functions

The multivariate Gaussian functions give hyperellipsoidal output densities:

$$G_g(\mathbf{x};\mathbf{t},\mathbf{b}) = e^{-D^2(\mathbf{x},\mathbf{t},\mathbf{b})} = \prod_{i=1}^{N} e^{-(x_i-t_i)^2/b_i^2} \tag{22}$$

Dispersions b_i may be interpreted as scaling factors in the distance function:

$$D^2(\mathbf{x};\mathbf{t},\mathbf{V}) = \sum_i V_i (x_i - t_i)^2 \tag{23}$$

with $V_i = 1/b_i^2$. Similar result is obtained by combining the sigmoidal output function (or any other logistic function) with the quadratic distance function, for example:

$$\begin{aligned} G_S(\mathbf{x};\mathbf{t},\mathbf{b}) &= 2\left(1 - \sigma(D^2(\mathbf{x};\mathbf{t},\mathbf{b}))\right) &(24)\\ &= \frac{2}{1+\prod_{i=1}^N e^{(x_i-t_i)^2/b_i^2}} = \frac{2}{1+e^{D^2(\mathbf{x},\mathbf{t},\mathbf{b})}} &(25)\end{aligned}$$

For N-dimensional input space each ellipsoidal unit uses $2N$ adaptive parameters. Taking the Mahalanobis distance function

$$D^2_M(\mathbf{x};\mathbf{t}) = \sum_{ij} (x_i - t_i)\Sigma^{-1}(x_i - t_i) \tag{26}$$

where Σ is the (symmetric) covariance matrix of $\mathbf{x} - \mathbf{t}$, rotation of hyperellipsoids is introduced. Treating the elements of covariance matrix as adaptive parameters is equivalent to the use of a general metric tensor in the distance function: $D^2_g(\mathbf{x};\mathbf{G};\mathbf{t}) = \sum_{i\geq j} G_{ij}(x_i - t_i)(x_i - t_i)$. The total number of parameters per each function becomes $N(N+3)/2$ and the constant density surfaces are given by general quadratic forms, i.e. they are ellipsoidal, parabolic or hyperbolic.

A single unit may also provide more complex densities if more general distance functions are used, but one should avoid too many parameters per one neural node. Simpler units giving approximately ellipsoidal densities are also useful, for example:

$$G_2(\mathbf{x};\mathbf{t},\mathbf{b}) = \prod_{i=1}^N \frac{1}{1+(x_i - t_i)^2/b_i^2} \tag{27}$$

This formula cannot be easily expressed in terms of an overall distance function. Using linear approximation for G_S (instead of a product) the squared distance function appears in the denominator:

$$G_3(\mathbf{x};\mathbf{t},\mathbf{b}) = \frac{1}{1+\sum_{i=1}^N (x_i - t_i)^2/b_i^2} = \frac{1}{1+D^2(\mathbf{x};\mathbf{t},\mathbf{b})} \tag{28}$$

These functions give hyperellipsoidal densities. Giraud et al. [12] used a fan-in function to create the *Lorentzian* response functions:

$$L(\mathbf{x};\mathbf{W}) = \frac{1}{1+I^2(\mathbf{x};\mathbf{W},\theta)} = \frac{1}{1+(\sum_{i=1}^N W_i x_i - \theta))^2} \tag{29}$$

Although G_S and L functions look similar they are in fact quite different: Lorentzian functions are not ellipsoidal, surfaces of constant density are in their case a window-type non-localized function, with the half-width equal to $1/\sqrt{\sum_i W_i^2}$.

A number of local training algorithms has been devised for local 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.

An interesting feature[2] of Gaussian functions G_g (Eq. 22) is that after a simple renormalization (Eq. 30) they become non-local and are equivalent to sigmoidal functions $\sigma(\mathbf{x};\mathbf{p})$, where $p_i = b_i^2/4t_i$:

$$G_R(\mathbf{x};\mathbf{t},\mathbf{b}) = \frac{G_g(\mathbf{x};\mathbf{t},\mathbf{b})}{G_g(\mathbf{x};\mathbf{t},\mathbf{b}) + G_g(\mathbf{x};-\mathbf{t},\mathbf{b})} = \frac{1}{1+e^{-4\sum_{i=1}^{N} x_i t_i/b_i^2}} \tag{30}$$

3.4 Universal transfer functions

Linear terms used to calculate $I(\mathbf{x};\mathbf{W},\theta)$ activations and quadratic terms used in Euclidean distance measures combined together create functions that for some parameters give localized, and for other parameters non-localized densities. Several functions of such kind have been proposed recently. Ridella et al. [33] use circular units in their Circular Backpropagation Networks. The output function is a standard sigmoid while the activation function contains one extra term:

$$I^c(\mathbf{x};\mathbf{W}) = W_0 + \sum_{i=1}^{N} W_i x_i + W_{N+1} \sum_{i=1}^{N} x_i^2 \tag{31}$$

and may also be presented in form of a distance function with:

$$\begin{aligned} I^c(\mathbf{x};\mathbf{W}) &= d_c(\mathbf{x};\mathbf{c}) = (||\mathbf{x}-\mathbf{c}||^2 - \theta) W_{N+1}; \\ c_i &= -W_i/2W_{N+1}; \quad \theta = \frac{1}{W_{N+1}} \left(\sum_{i=1}^{N} \frac{W_i^2}{4W_{N+1}^2} - W_0 \right) \end{aligned} \tag{32}$$

Ridella et al. [33] obtained very good results using these units in the standard backpropagation network and proved that in many ways circular units provide an optimal solution in classification problems. Different type of circular units have been used by Kirby and Miranda [20]. In their implementation two sigmoidal units are coupled together and their output is restricted to lie on a unit circle.

Dorffner [6] proposed conic section transfer functions as a unified framework for MLP and RBF networks. Straight lines and ellipses are special cases of conic sections. From geometrical considerations Dorffner proposes a combination of fan-in and distance activation functions:

$$\begin{aligned} C(\mathbf{x};\mathbf{W},\mathbf{t},\omega) &= I(\mathbf{x}-\mathbf{t};\mathbf{W}) + \omega D(\mathbf{x}-\mathbf{t}) \\ &= \sum_{i=1}^{N+1} W_i(X_i - t_i) + \omega \sqrt{\sum_{i=1}^{N+1} (x_i - t_i)^2} \end{aligned} \tag{33}$$

This activation is then composed with the standard sigmoidal function to produce the conical transfer function. From our previous discussion it should be clear that many other combinations of fan-in and distance functions could also serve as universal transfer functions. For example, $\exp(\alpha I^2 - \beta D^2)$ or the approximated Gaussian combined with the Lorentizan function also provide an interesting universal transfer function:

[2] W.D. is indebted to Igor Grabiec for pointing this out in a private discussion

$$C_{GL}(\mathbf{x};\mathbf{W},\mathbf{t},\alpha,\theta) = \frac{1}{1+\alpha I^2(\mathbf{x};\mathbf{W},\theta)+\beta D^2(\mathbf{x};\mathbf{t})} \tag{34}$$

For simplicity we may assume that $\beta = 1-\alpha$. The α parameter scales the relative importance of the linear, non-localized terms. The number of adaptive parameters in this case is equal to $2N+1$ (no scaling factors in distance function) or $3N+1$ (separate distance scaling factors for each dimensions). Unfortunately universal functions are nonseparable.

3.5 Biradial functions

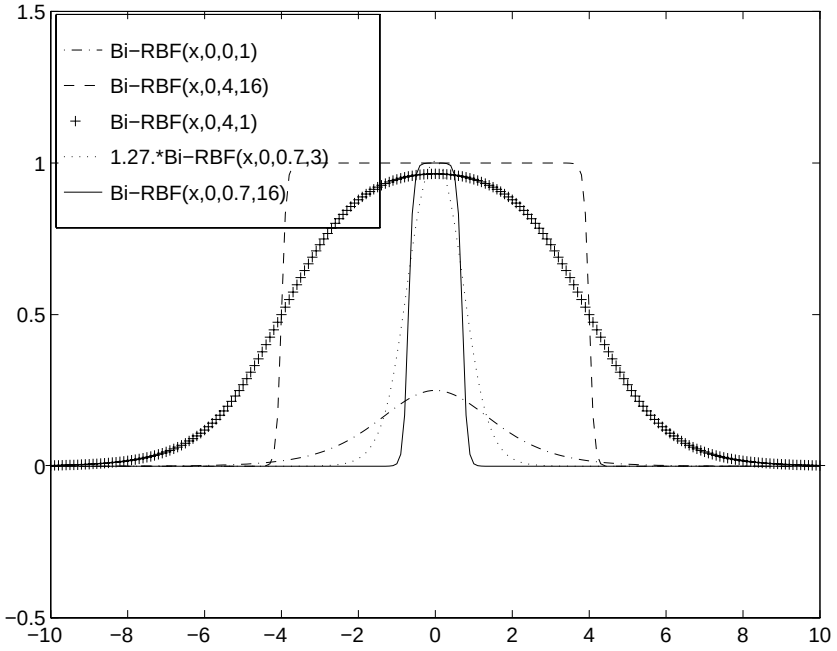

Figure 4: A few shapes of the biradial functions in one dimension.

Sigmoidal functions may be combined into a “window" type localized functions in several ways. Perhaps the simplest is to take the difference of two sigmoids, $\sigma(x)-\sigma(x-\theta)$. One may also use products of pairs of sigmoidal functions $\sigma(x)(1-\sigma(x))$ for each dimension. This type of transfer functions are very flexible, producing decision regions with convex shapes, suitable for classification. Product of N pairs of sigmoids has the following general form:

$$Bi(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) = \prod_{i=1}^{N} \sigma(e^{s_i}\cdot(x_i - t_i + e^{b_i}))(1-\sigma(e^{s_i}\cdot(x_i - t_i - e^{b_i}))) \tag{35}$$

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 $\mathbf{s}$. Radial basis functions are defined relatively to only one center $||x-t||$. Here

two centers are used, $t_i + e^{b_i}$ and $t_i - e^{b_i}$, therefore we call these functions biradial. Product form leads to well-localized convex densities of biradial functions.

The number of adjustable parameters per processing unit is in this case $3N$. Dimensionality reduction is possible as in the *Gaussian bar case*, but more flexible density shapes are obtained, 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 parameters to prevent oscillations during the learning procedure (learning becomes more stable).

Localized biradial functions may be extended to the semi-localized universal transfer functions by adding two parameters:

$$SBi((\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) = \prod_{i=1}^{N}(\alpha+\sigma(e^{s_i}\cdot(x_i-t_i+e^{b_i})))(1-\beta\sigma(e^{s_i}\cdot(x_i-t_i-e^{b_i}))) \quad (36)$$

This function does not vanish for large $|x|$, for $\alpha = 0$, $\beta = 1$ it is identical to the biradial localized functions while for $\alpha = \beta = 0$ each component under the product turns into the usual sigmoidal function. For each unit semi-local functions SBi have $3N+2$ parameters or $5N$ parameters (if different α_i and β_i are used in each dimension).

4 Neural networks with biradial transfer functions

As far as we know biradial functions as well as universal functions C_{GL} Eq. (34) have never been tested before in the neural network context. We have performed tests of biradial transfer functions for classification and approximation problems with two different neural network architectures. To test the difference in performance between the standard sigmoidal and biradial transfer functions for classification we have used the modified Stuttgart Neural Networks Simulator (SNNS) [17]. Backpropagation formulas for the biradial transfer functions were derived and implemented in the RBF package. We have also modified RBF to work not only with radial, but also with sigmoidal transfer function.

The two-spiral classification benchmark[3] is a difficult test frequently used for backpropagation networks. The number of data points used for training is 196, points are divided into two classes (represented by the dark and light areas in Figure 5). Three RBF-type networks of identical structure were trained using the two-spiral data. The same initialization procedure was used with the Gaussian, sigmoidal and biradial transfer functions. The number of network nodes was set to 100 (about half of the number of the training vectors) and each network was trained for 2000 epochs.

In Figure 6 convergence of errors during training of the RBF network using Gaussian transfer functions (Eq. 14, with the same dispersion in each dimension, but optimized for each node), sigmoidal functions (Eq. 2, with the same dispersion and slope in each dimension, also optimized for each node) and biradial transfer functions (Eq. 35, all dispersions and slopes optimized by the learning procedure) is presented. The network based on biradial transfer functions not only learns faster (Fig. 6) but also generalizes better (Fig. 5). It is interesting to note that the sigmoidal functions used in the RBF-type of networks performed much better than Gaussian functions. The two-spiral problem is easy for the Gaussian RBF network if the number of nodes is equal to the number of training vectors. If the number of nodes is restricted Gaussian functions are not sufficiently flexible to represent the density accurately.

[3] These benchmark data is stored at: http://www.cs.cmu.edu/afs/cs/project/connect/bench/.

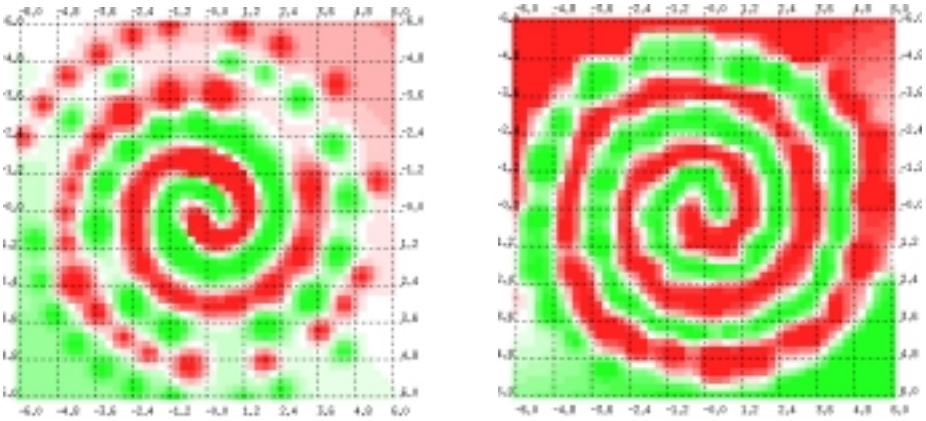

Figure 5: Results for the two spiral classification problem solved with the Gaussian (left figure) and biradial (right figure) transfer functions.

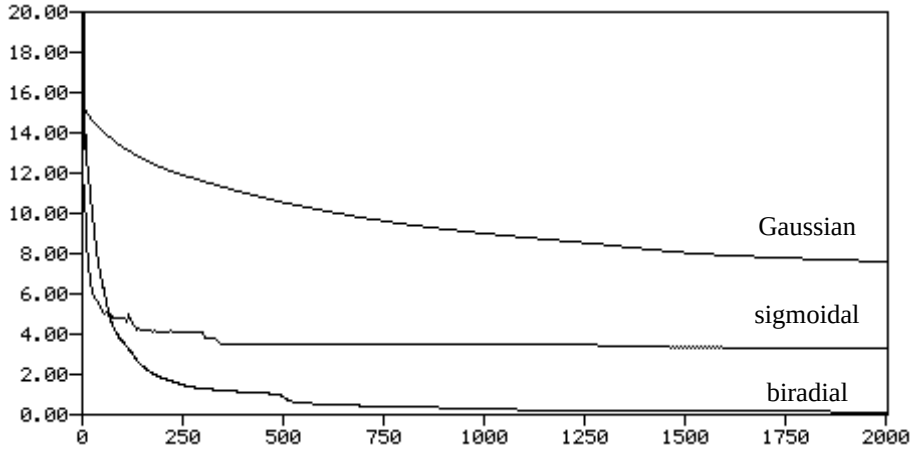

Figure 6: Comparison of the summed squared errors for different transfer functions: Gaussian (Eq. 14), sigmoidal (Eq. 2) and biradial function (Eq. 35) used in the same RBF net during 2000 epochs.

Our second benchmark problem concerns approximation rather than classification. Approximation of Sugeno function [35] $f(x,y,z) = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2$ was attempted using Gaussian and biradial transfer functions. Although this function is frequently used for testing the approximation capabilities of adaptive systems there is no standard procedure to select the training points and thus the results are rather hard to compare. Here for training 216 points from $[1,6]$ section and 125 points for testing from $[1.5,5.5]$ section were randomly chosen. Since our goal is to test the usefulness of the biradial functions the results of computations made by the IncNet neural network [19] with Gaussian and biradial functions are compared. IncNet is a network with statistical control of growing and pruning of neurons in hidden layer in RBF-like structure network. All tests were performed using the same initial parameters.

Two learning processes were pursued for 4000 iteration[4] Although it is possible to obtain smaller RMS error by increasing the number of iterations and changing other

[4]One iteration is a single update of parameters in which only one pattern is presented for learning.

parameters used in control of learning it will not change the qualitative difference of our results. The change of RMS errors in the training process is presented in Figure 7. Learning using Gaussian functions is unstable, the network is unable to build suitable *landscape* to approximate the function. The IncNet network gives clearly better results using biradial transfer functions than Gaussian functions.

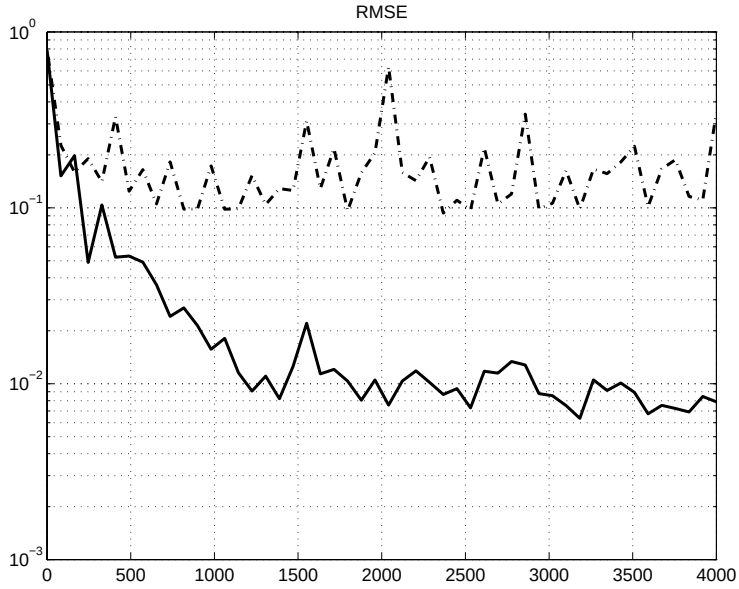

Figure 7: Comparison of RMS errors obtained by the IncNet network using biradial (solid line) and Gaussian (dashdot line) functions.

5 Discussion and Possible Extensions

We have presented an overview of different transfer functions used in neural network models and presented several new functions suitable for this purpose. From the geometrical point of view learning requires approximation of complicated probability densities. In the process of density estimation by neural networks flexible transfer functions are as important as good architectures and learning procedures. Small number of network parameters should allow for maximum flexibility. Universal (semi-localized) functions, such as the circular, conical, biradial or simplified Lorentzian/Gaussian functions lead to more compact networks that learn faster. These functions unify the distance-based, localized paradigm using terms quadratic in inputs, with the non-local approximations based on discriminant functions that use only the linear terms.

An important advantage of the biradial functions comes from their separability. Sigmoidal functions are not separable and among radial basis functions only Gaussians are separable. Separability enables analysis of each dimension or a subspace of the input data independently. In other words one can forget some of the input features and work in the remaining subspace. This is very important in classification when some of the features are missing.

Biradial transfer functions may also be used for *logical rule extraction* using FSM density estimation network. Logical interpretation of the function realized by neural

network is possible if instead of hyperellipsoidal densities cuboidal densities are used. In case of sigmoidal and biradial transfer functions sufficiently big values of the slopes are needed, changing graded sigmoidal functions into step functions and biradial functions into cuboidal (rectangular) functions. There are several ways to enforce large slopes of the transfer functions. The network may be trained with modified error function, for example:

$$E_{new} = E_{old} + \gamma \sum_i 1/s_i^2 \tag{37}$$

Modification of the error function may also be done after the training process is completed, with subsequent retraining to maximize the slopes with minimal change of the network parameters. The “window" for irrelevant inputs becomes broad and when it covers all the data the links to these inputs are removed. Using these ideas we have obtained very good results in applications to rule extraction from data (Duch, Adamczak and Grąbczewski, in preparation).

The biradial functions proposed and tested in this paper contain $3N$ parameters per one unit and are quite flexible in representing various probability densities. Semi-biradial functions provide local and non-local units in one network. Next step towards even greater flexibility requires individual rotation of densities provided by each unit. Of course one can introduce a rotation matrix operating on the inputs $\mathbf{Rx}$, but in practice it is very hard to parametrize this $N \times N$ matrix with $N-1$ independent angles (for example, Euler's angles) and calculate the derivatives necessary for the backpropagation procedure. We have found two ways to obtain rotated densities in all dimensions using transfer functions with just N additional parameters per neuron. In the first approach product form of the combination of sigmoids is used

$$\begin{aligned} C_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{R}) &= \prod_i \Big(\sigma(\mathbf{R}_i\mathbf{x}+t_i) - \sigma(\mathbf{R}_i\mathbf{x}+t_i')\Big) \\ SC_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{p},\mathbf{r},\mathbf{R}) &= \prod_i \Big(p_i\cdot\sigma(\mathbf{R}_i\mathbf{x}+t_i) + r_i\cdot\sigma(\mathbf{R}_i\mathbf{x}+t_i')\Big) \end{aligned} \tag{38}$$

where $\mathbf{R}_i$ is the i-th row of the rotation matrix $\mathbf{R}$ with the following structure:

$$\mathbf{R} = \begin{bmatrix} s_1 & \alpha_1 & 0 & & \cdots & 0 \\ 0 & s_2 & \alpha_2 & 0 & & \\ & & & & & \\ \vdots & & & \ddots & & \vdots \\ & & & & s_{N-1} & \alpha_{N-1} \\ 0 & \cdots & & & 0 & s_N \end{bmatrix} \tag{39}$$

If $p_i = 1$ and $r_i = -1$ then SC_P function is localized and gives similar densities as the biradial functions (except for rotation). Choosing other values for the p_i and r_i parameters non-local transfer functions are created.

In the second approach the density is created by the sum of a “window-type" combinations of sigmoids $L(x;t,t') = \sigma(x+t) - \sigma(x+t')$ in $N-1$ dimensions and a combination rotated by a vector $\mathbf{K}$:

$$C_K(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{W},\mathbf{K}) = \sum_{i=1}^{N-1} W_i L(x_i,t_i,t_i') + W_N L(\mathbf{Kx},t,t') \tag{40}$$

The last density is perpendicular to the $\mathbf{K}$ vector. Treating $C_K(\cdot)$ as the activation function and using sigmoidal output function with a proper threshold leaves only the densities in the direction perpendicular to $\mathbf{K}$. An alternative is to use the product form:

$$C_{PK}(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{K}) = L(\mathbf{K}\mathbf{x},t,t') \prod_{i=1}^{N-1} L(x_i,t_i,t_i') \tag{41}$$

as the transfer function – the output sigmoid is not needed in this case. Rotation adds only $N-1$ parameters for $C_P(\cdot)$ function and N parameters for $C_K(\cdot)$ function.

So far we have not seen any adaptive systems using such generalized transfer functions. There is an obvious tradeoff between the flexibility of the processing units increasing with the number of adjustable parameters and the complexity of the training process of the whole network. Biradial and rotated transfer functions ($C_P(\cdot)$, $C_S(\cdot)$) are flexible but still rather simple, therefore we intend to use them also in the FSM [7] and other networks.

Although the importance of density estimation seems rather obvious the value of research on the transfer functions is frequently overlooked. We believe that the point of view presented in this paper is very fruitful and should be pursued.

References

[1] John Allison. Multiquadratic radial basis functions for representing multidimensional high energy physics data. *Computer Physics Communications*, 77:377–395, 1993.

[2] Chris Bishop. Improving the generalization properties of radial basis function neural networks. *Neural Computation*, 3:579–588, 1991.

[3] L. Bottou and V. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

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

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

[6] Georg Dorffner. A unified framework for MLPs and RBNFs: Introducing conic section function networks. *Cybernetics and Systems*, 25(4), 1994.

[7] Włodzisław Duch and G. H. F. Diercksen. Feature space mapping as a universal adaptive system. *Computer Physics Communications*, 87:341–371, 5 1994.

[8] R. Durbin and D. E. Rumelhart. Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, 1:133–142, 1989.

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

[10] R. Franke. Scattered data interpolation: test of some methods. *Math Computation*, 38:181–200, 1982.

[11] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1972.

[12] B. G. Giraud, A. Lapedes, L. C. Liu, and J. C. Lemm. Lorentzian neural nets. *Neural Networks*, 8:757–767, 1995.

[13] E. Hartman and J. D. Keeler. Predicting the future: Advantages of semilocal units. *Neural Computation*, 3(4):566–578, 1991.

[14] E. J. Hartman, J. D. Keeler, and J. M. Kowalski. Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation*, 2(2):210–215, 1990.

[15] Simon Haykin. *Neural Networks - A Comprehensive Foundation*. IEEE Press, 1994.

[16] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[17] Institute of Parallel and Distributed High-Performance Systems (IPVR). *Stuttgart Neural Networks Simulator*. http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html.

[18] V. Kadirkamanathan and M. Niranjan. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5(6):954–975, 1993.

[19] Visakan Kadirkamanathan and Norbert Jankowski. Rbf-like neural networks statistical control of growing and pruning. in preparation, 1997.

[20] M. J. Kirby and R. Miranda. Circular nodes in neural networks. *Neural Computaitons*, 8:390–402, 1996.

[21] H. Leung and S. Haykin. Rational neural networks. *Neural Computation*, 5, 1993.

[22] D. Lowe. Adaptive radial basis function nonlinearities, and the problem of generalisation. In *1st IEE International Conference on Artifical Neural Networks*, pages 171–175, 1989.

[23] D. Lowe. On the iterative inversion of rbf networks: Astatistical interpretation. In *2nd IEE International Conference on Artifical Neural Networks*, pages 29–33, 1991.

[24] D. Lowe. Novel “topographic" nonlinear. In *3rd IEE International Conference on Artifical Neural Networks*, pages 29–33, 1993.

[25] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[26] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, pages 281–294, 1989.

[27] M. Morita. Memory and learning of sequential pattern by nonmonotone neural networks. *Neural Networks*, 9:1477–1489, 1996.

[28] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading, MA, 1989.

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

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

[31] T. Poggio and F. Girosi. Network for approximation and learning. *Proc. IEEE*, 78(9):1481–1497, September 1990.

[32] M. J. D. Powell. Radial basis functions for multivariable interpolation: A review. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation of Functions and Data*, pages 143–167. Oxford University Press, 1987.

[33] S. Ridella, S. Rovetta, and R. Zunino. Circular backpropagation netwoprks for classification. *IEEE Transaction on Neural Networks*, 8:84–97, 1997.

[34] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[35] M. Sugeno and G. T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28:13–33, 1988.