Survey of Neural Transfer Functions

Włodzisław Duch & Norbert Jankowski
Department of Computer Methods, Nicholas Copernicus University,
ul. Grudziądzka 5, 87-100 Toruń, Poland
http://www.phys.uni.torun.pl/kmk

Abstract

The choice of transfer functions may strongly influence complexity and performance of neural networks. Although sigmoidal transfer functions are the most common there is no *a priori* reason why models based on such functions should always provide optimal decision borders. A large number of alternative transfer functions has been described in the literature. A taxonomy of activation and output functions is proposed, and advantages of various non-local and local neural transfer functions are discussed. Several less-known types of transfer functions and new combinations of activation/output functions are described. Universal transfer functions, parametrized to change from localized to delocalized type, are of greatest interest. Other types of neural transfer functions discussed here include functions with activations based on non-Euclidean distance measures, bicentral functions, formed from products or linear combinations of pairs of sigmoids, and extensions of such functions making rotations of localized decision borders in highly dimensional spaces practical. Nonlinear input preprocessing techniques are briefly described, offering an alternative way to change the shapes of decision borders.

Keywords: Neural networks, adaptive systems, local learning, transfer functions, activation functions, minimal distance methods.

1 INTRODUCTION

Adaptive systems of the Artificial Neural Network (ANN) [1, 2, 3] 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 $Y = F_W(X)$. From the probabilistic point of view [2, 3] adaptive systems should approximate the density of joint probability $p(X,Y)$ or the posterior probability $p(Y|X)$ of input-output values. Flexibility of contours of transfer functions used for estimation of decision borders is strongly correlated with the number of functions (and thus with the number of adaptive parameters available for training) necessary to model complex shapes of decision borders. The current emphasis in neural network research is on learning algorithms and architectures, neglecting the importance of transfer functions. In approximation theory many functions are used (cf. [4]), while neural network simulators use almost exclusively sigmoidal or Gaussian functions. This paper presents a survey of transfer functions suitable for neural networks in an attempt to show the potential hidden in their selection.

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

[0] Support by the Polish Committee for Scientific Research, grant 8T11F 014 14, is gratefully acknowledged. Updates, corrections, and comments should be sent to W. Duch at duch@phys.uni.torun.pl.

of the important characteristics being the transfer functions performed by each neuron. The first attempts to build neural network models was based on logical networks [5], or threshold devices performing step functions. These step functions were generalized in a natural way to functions of sigmoidal shape. Neural networks with single hidden layer using sigmoidal functions are universal approximators [6, 7], i.e. they can approximate an arbitrary continuous function on a compact domain with arbitrary precision given sufficient number of neurons. These mathematical results do not mean that sigmoidal functions provide always an optimal choice or that a good neural approximation is easy to find. For some datasets a large (and hard to train) network using sigmoidal functions may be needed for tasks that could be solved with a small (and easy to train) network using other transfer functions. Networks with neurons that give Gaussian outputs instead of sigmoidal outputs are also universal approximators [8, 9].

A new type of transfer functions, called *gaussian bars*, has been proposed by Hartman and Keeler [10]. In the *functional link* networks of Pao [11] 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 [12]. In the *conic section* function networks Dorffner [13] 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.* [14]. These and many other papers surveyed here show that the choice of transfer functions is considered by some experts to be 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 [2, 3]. None of the functions mentioned above is flexible enough to describe, using a small number of adaptive parameters, an arbitrarily shaped decision borders in multidimensional input space. Statisticians prefer to test their methods on artificial data [15, 16]. It is easy to notice that some data distributions are handled easily using localized functions, for example Gaussians, while other data may be handled in an easier way if non-local functions are used, for example sigmoidal functions with weighted activations. Following statisticians, consider [15] a simple classification problem in N dimensions, with spherical distribution of vectors belonging to the first class that should be distinguished from vectors belonging to the second-class, lying outside the unit sphere. A single neuron performing multivariate Gaussian function with $2N$ adaptive parameters (specifying the center and dispersions in each dimension) is sufficient for this job and the training process is quite simple. Many hyperplanes provided by sigmoidal functions are needed to approximate spherical decision borders. The simplest approximation using the standard multilayer perceptron (MLP) architecture that captures any bound region of N-dimensional space requires construction of a simplex using N sigmoids and additional neuron to smooth the output of a combination of these neurons, so at least N^2+N parameters are needed in this case, making the training process much more difficult. On the other hand if data vectors belonging to the first class are taken from the corner of the coordinate system bound by the $(1,1,...,1)$ plane a single sigmoidal function with $N+1$ parameters is sufficient for perfect classification while Gaussian approximation will be quite difficult. A poor approximation may use one Gaussian in the center of the region and $N+1$ Gaussians in the corners, using $2N(N+2)$ adaptive parameters and making the learning harder than with the hyperplanar decision borders.

One may easily create more complicated examples with more classes between concentric spheres of growing radii or with series of hyperplanes passing through $(m,m,...,m)$ points. Improved learning algorithms or network architectures will not change the relative complexity of solutions as long as the decision borders provided by the transfer functions remain spherical (as in the first example) or planar (as in the second example). Artificial examples that are favorable for other types of functions are also easy to construct. For the real-world data one may compare the best results obtained using the Radial Basis Function (RBF) networks with Gaussian functions and the MLP networks with sigmoidal functions. According to the Statlog report [17] comparing many classification methods on 22 real-world datasets results of RBF are not similar to the results of MLP. In some cases crossvalidation errors were twice as large using MLP than RBF (for example for the DNA dataset RBF gave 4.1% of error and was the best of all methods while MLP gave 8.8 % of errors and was on 12-th position, while for the Belgian Power data the situation is reversed, with MLP giving 1.7% of errors and RBF 3.4%). Although one may always argue that better initialization, learning and architectures will reduce the difference one should admit that the differences exist, if not in the absolute accuracy or generalization capability, than in the ease of finding good solutions. At least part of the difference in performance of the two neural models used above should be attributed to different shapes of decision borders (hyperplane and ellipsoidal) provided by their transfer functions. Amari and Wu (Neural Networks, in press 1999) give examples of significant improvements for Support Vector Machine classifiers due to modifications of the kernel functions.

Viewing the problem of learning from geometrical point of view functions performed by neural nodes should enable tessellation of the input space in the most flexible way using a small number of adaptive parameters. Implications of this fact has not yet been fully appreciated by many researchers, but the very fact that we have already found quite a few papers describing the use of different transfer functions shows that the issue is interesting for many researchers and therefore worth surveying. In this paper we systematically investigate various functions suitable as the transfer functions for neural networks. We do not attempt to cover all activities related to transfer functions. Good transfer functions approximating the biologically faithful neural response functions have not been found yet and the very concept is rather difficult to define. Anderson [18] justifies sigmoidal functions for motoneurons but a transition from spiking neurons of associative cortex to model neurons using continuous transfer functions is certainly not so simple (for a theoretical introduction to the capabilities of spiking neurons see [19]). Very interesting attempts to build analogue neurons or hardware models of such neurons are also beyond the scope of this review [20, 21, 22]. To keep the paper rather short, various transfer functions used only in associative memory models, such as nonmonotonic functions [23, 24, 25, 26, 27], periodic functions [28, 28, 29, 30] or chaotic neurons [31, 32], have also been omitted here, although they may actually be more faithful to neurobiology, may help to increase the capacity of associative memories and avoid spurious local minima of the neural network error functions. Fuzzy neural networks [33] use specific transfer functions and are also omitted here. Neural models using complex transfer functions (cf. [34]) are also omitted. There is a vast statistical literature on approximation, discrimination, kernel regression and locally weighted regression [35], support vector machines [36, 37, 38] and other subjects interesting in the context of neural networks, but this survey is restricted to systematic investigation of transfer functions suitable for feedforward and some recurrent networks.

Information about various transfer functions is scattered in the literature and has not been reviewed so far. Many of the functions described in this paper have never been used in practice yet and little is known about their relative merits. We have tried to provide a taxonomy of activation and output functions (shown in Fig. 2 and Fig. 4) and show how they are combined in transfer functions. As with most taxonomies it is not perfect and may be done in several other ways. Most transfer functions admit natural separation into the output and activation functions, but for a few transfer functions it is rather difficult and arbitrary. Still we have found the taxonomy presented here rather helpful, allowing to discover new useful combinations of activation and output functions.

In the next section general issues related to transfer functions are described, including a discussion of the hard and soft versions of transfer functions. The third section contains detailed presentation of activation functions, including discussion of distance functions, and the fourth section presents various output functions. Comparison of transfer functions is made in the fifth section. In the next section non-linear transformation of input data providing complex decision borders is presented as an alternative to selection of transfer functions. Unfortunately comparison of results that may be obtained using different transfer functions is very difficult. Several groups of transfer functions could be used in networks with identical architecture, initialization and training. Although such comparison is certainly worthwhile it has not yet been attempted and would require a large-scale effort going far beyond our survey. Many functions are used in different models, therefore a direct comparison is not always possible. More remarks on this issue are given in the discussion closing this paper.

2 GENERAL ISSUES

Two functions determine the way signals are processed by neurons. *The activation function* determines the total signal a neuron receives. In this section a fan-in function, i.e. a linear combination of the incoming signals, is used, but in the next section other possibilities are presented. 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 $I_i(\mathbf{x})$ is

$$I_i(\mathbf{x}) = \sum_{j=0}^{N} W_{ij} x_j \tag{1}$$

where $W_{i,0} = \theta$ (threshold) and $x_0 = 1$.

The value of the activation function is usually scalar and the arguments are vectors. The second function determining neuron's signal processing is *the output function* $o(I)$, operating on scalar activations and returning scalar values. Typically a squashing function is used to keep the output values within specified bounds.

These two functions together determine the values of the neuron outgoing signals. The composition of the activation and the output function is called the *transfer function* $o(I(\mathbf{x}))$. The transfer function is defined in the N-dimensional *input space*, called also *the parameter space*. For some transfer functions there is no natural division between activation and output functions. The transfer function is local if its values are significantly different from zero (i.e. $|o(I(\mathbf{x}))| > \varepsilon$ for some small ε) in a finite area of the input space; otherwise the function is non-local. Locality depends both on the activation and transfer function.

In neural models 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 linear functions are frequently 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 [5] were based on the logical processing elements. The output function of the logical elements is of *the step function* type, and is also known as the Heaviside $\Theta(I;\theta)$ function:

$$\Theta(I;\theta) = \begin{cases} 1 & I > \theta \\ 0 & I \leq \theta \end{cases} \tag{2}$$

i.e. 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. Decision borders provided by logical neurons are hyperplanes rotated by the W_{ij} coefficients. Networks of logical neurons divide the input space into polyhedral areas.

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:

$$\varsigma(I) = y_i \quad \text{for} \quad \theta_i \leq I < \theta_{i+1} \tag{3}$$

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:

$$s_l(I;\theta_1,\theta_2) = \begin{cases} 0 & I \leq \theta_1, \\ (I-\theta_1)/(\theta_2-\theta_1) & \theta_1 < I \leq \theta_2 \\ 1 & I > \theta_2 \end{cases} \tag{4}$$

These functions have discontinuous derivatives, preventing the use of gradient-based error minimization training procedures. Therefore they were later generalized to *the logistic output* functions, leading to the *graded response neurons*, used most often in the literature (see Fig. 1):

$$\sigma(I/s) = \frac{1}{1+e^{-I/s}} \tag{5}$$

The constant s determines the slope of the logistic function around the linear part. There are many functions similar in shape to the logistic function, forming a broad class of *sigmoidal functions*. In the hard limit, when the slope

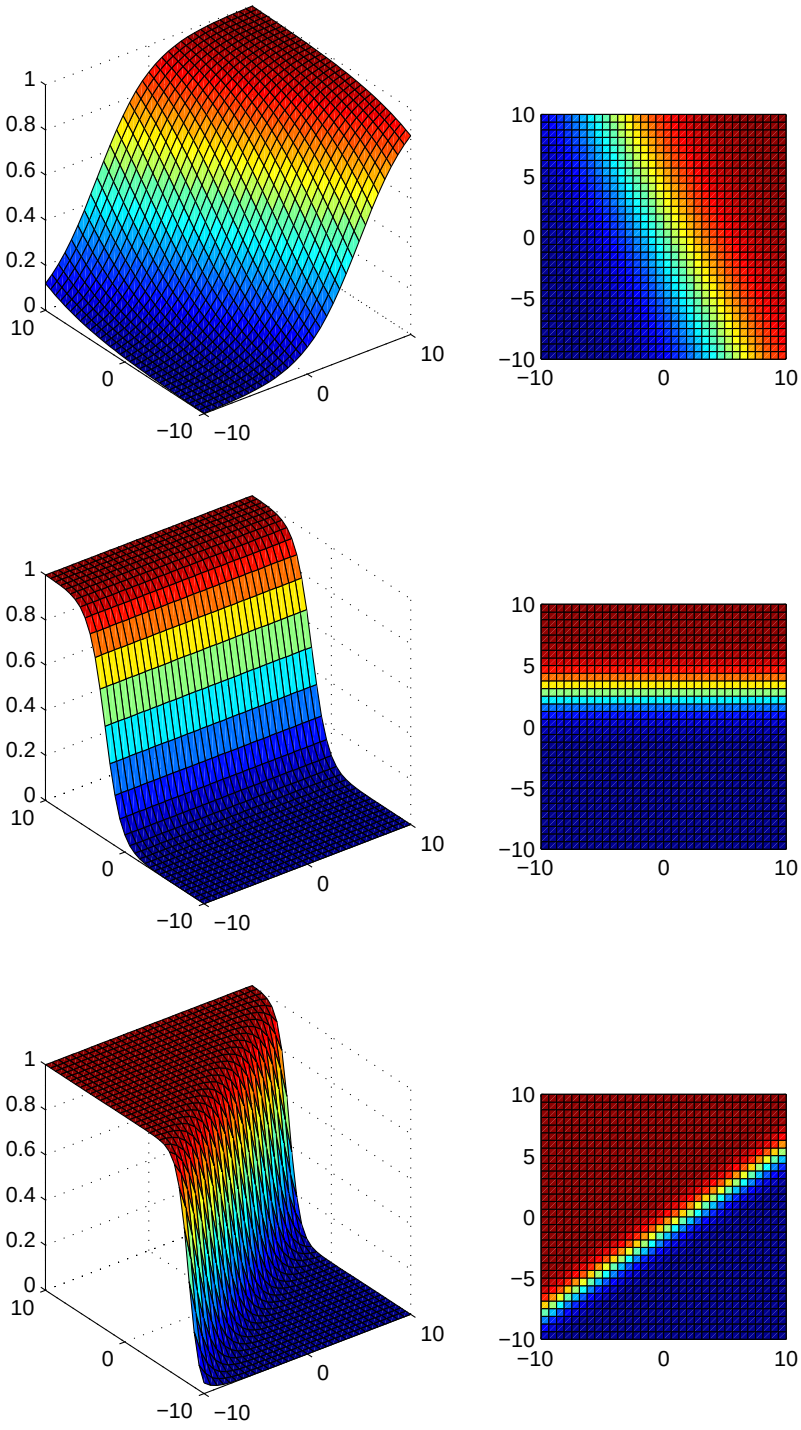

Figure 1: Logistic functions with inner product activations.

of these S-shaped functions becomes infinite ($s \to 0$), their derivatives become discontinuous and all these functions become step functions.

Combining sigmoidal output functions with the fan-in activation given by (Eq. 1) *sigmoidal transfer functions* are obtained. These transfer functions are non-local, but combining sigmoidal output functions with some other activations may result in localized transfer functions (cf. Eq. 62–65).

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. Except for some neurobiological inspirations sigmoids may also be justified from a statistical point of view [2, 39]. Consider a classification problem in N dimensions with two classes described by Gaussian distributions with equal covariance matrices (more general *exponential family* of distributions may be used [2, 39]):

$$p(\mathbf{x}|C_k) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\overline{\mathbf{x}}_k)^T\Sigma^{-1}(\mathbf{x}-\overline{\mathbf{x}}_k)\right\} \tag{6}$$

Using Bayes' theorem the posterior probability for the first class is:

$$p(C_1|\mathbf{x}) = \frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_1)P(C_1)+p(\mathbf{x}|C_2)P(C_2)} = \frac{1}{1+\exp(-y(\mathbf{x}))} \tag{7}$$

where $P(C_k)$ are *a priori* class probabilities and the function $y(\mathbf{x})$ is:

$$y(\mathbf{x}) = \ln\frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_2)P(C_2)} \tag{8}$$

Of course $p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x})$. Thus the Bayesian approach for the two-class problems leads to the logistic output functions with rather complex activation functions. Such functions are used in statistics in logistic discrimination [40]. For more than two classes normalized exponential functions (called also softmax functions) are obtained by the same reasoning:

$$p(C_k|\mathbf{x}) = \frac{\exp(y_k(\mathbf{x}))}{\sum_i \exp(y_i(\mathbf{x}))} \tag{9}$$

These normalized exponential functions may be interpreted as probabilities.

An interesting alternative explanation [41] of the usefulness of sigmoidal functions with weighted activation (i.e. sigmoidal transfer functions) is given below. Since input values result usually from observations which are not quite accurate, instead of a number y a Gaussian distribution $G_y = G(y;\overline{y},s_y)$ centered around $\overline{y}$ with dispersion s_y should be given. This distribution may be treated as a membership function of a fuzzy number G_y [33]. The cumulative distribution function is:

$$p(x-\overline{y}) = \int_{-\infty}^{x} G(y;\overline{y},s_y)dy = \frac{1}{2}\left[1+\mathrm{erf}\left(\frac{x-\overline{y}}{s_y\sqrt{2}}\right)\right] \approx \sigma\left(\frac{x-\overline{y}}{T}\right) \tag{10}$$

where erf is the error function and $T = \sqrt{2}s_y/2.4$. The acccuracy of this approximation is better than 0.02 for all x. The cummulative distribution $p(x-\overline{y})$ may be interpreted as the probability that a certain decision rule $R_x(z) =$True iff $z \le x$ is true, i.e. $p(R_x|G_y) = p(x-\overline{y})$. The Gaussian assumption for the uncertainty of inputs is equivalent to the *soft trapeziodal* membership functions of the logical rules used with the sharply defined inputs. On the other hand starting with sigmoidal functions instead of the erf function is equivalent to the assumption that the measurement uncertainties are given by $\sigma((x-\overline{y})/T)(1-\sigma((x-\overline{y})/T))$, approximating a Gaussian function within a few percent.

In practical applications of neural networks biological inspirations may not be so important as inspirations from approximation theory, probability theory, statistics or pattern recognition. This understanding led to neural models based on the radial basis functions, popular in approximation theory [42, 43]. Slowly other types of transfer functions were introduced, but systematic research of this aspect of neural models has been missing. In the next section we have tried to systematize our knowledge of the activation functions.

3 ACTIVATION FUNCTIONS

Weighted activation, called also the fan-in activation (Eq. 1), is used in neural models not only due to its biological inspirations, but because the contours of constant value $I(\mathbf{x}) = const$ define hyperplanes. Statistical methods of classification may be divided into two broad groups: methods based on discrimination, using hyperplanes or other hypersurfaces for tessellation of the input space, and methods based on clusterization, in which similarities are calculated using some kind of a distance measure. Therefore it seems that we have three main choices for activation functions:

- The **inner product** $I(\mathbf{x};\mathbf{w}) \propto \mathbf{w}^T \cdot \mathbf{x}$ (as in the MLP networks).
- The **distance based** activation, or more general similarity functions, $D(\mathbf{x};\mathbf{t}) \propto ||\mathbf{x}-\mathbf{t}||$, used to calculate similarity of $\mathbf{x}$ to a prototype vector $\mathbf{t}$.
- A **combination** of the two activations, $A(\mathbf{x};\mathbf{w},\mathbf{t}) \propto \alpha\, \mathbf{w}^T \cdot \mathbf{x} + \beta\, ||\mathbf{x}-\mathbf{t}||$,

In each case we may use either the final scalar value of activation, or use the vector components of this activation, for example using the distance form we usually take the scalar $D(\mathbf{x},\mathbf{t})$, but for some output functions we may also use the vector components $D_i(x_i,t_i) \propto (x_i-t_i)^2$, for example:

$$D_i(x_i,t_i,b_i) = (x_i-t_i)^2/b_i^2 \tag{11}$$

The square of the activation function is a quadratic form. Treating all coefficients of this form as independent and transforming it into canonical form:

$$I^2(\mathbf{x};\mathbf{w}) \sim D^2(\mathbf{x};\mathbf{t},\mathbf{a}) = \sum_i^N a_i (x_i' - t_i)^2 \tag{12}$$

where the new variables x_i' are linear combinations of the original variables x_i, leads to the pseudo-Euclidean distance function. If all parameters a_i are positive, $a_i = 1/b_i^2$, then a Euclidean distance function is obtained, providing hyperellipsoidal contours of constant values. Squared fan-in activation function is used in the *Lorentzian* transfer functions (Eq. 71, Fig. 14). Lorentzian functions are not ellipsoidal, surfaces of constant density are in their case a window-type non-localized function.

3.1 Distance Based Activation — Distance Functions.

Although activation is almost always associated with weighted combination of inputs it may also be based on evaluation of similarity of the incoming vectors to some reference or prototype vectors. There is nothing special about the Euclidean distance function used to compute distances in many radial basis and other functions. The Euclidean distance has a natural generalization in form of the Minkovsky's distance function:

$$D_M(\mathbf{x},\mathbf{y};\alpha) = \left(\sum_{i=1}^N |x_i - y_i|^\alpha \right)^{1/\alpha} \tag{13}$$

Euclidean and Manhattan distances are of course special cases of Minkovsky's metric function with $\alpha = 2$ and $\alpha = 1$ respectively. Minkovsky's distance with the scaling factors is a further generalization:

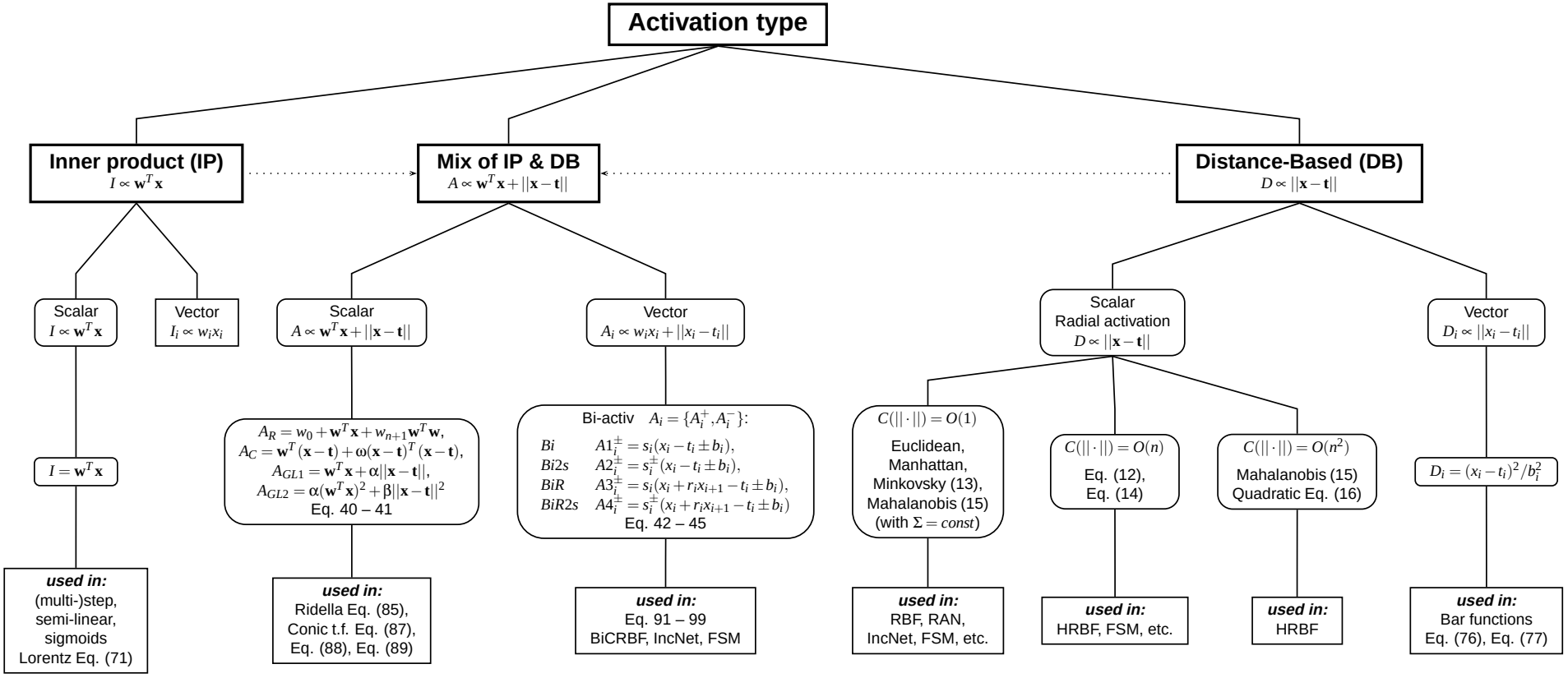

Figure 2: Taxonomy of activation functions. $C(||\cdot||)$ is the number of adaptive parameters of $||\cdot||$ norm.

$$D_{Mb}(\mathbf{x},\mathbf{y};\mathbf{b})^{\alpha} = \sum_{i}^{N} d(x_i,y_i)^{\alpha}/b_i \tag{14}$$

The $d(\cdot)$ function is used to estimate similarity at the feature level and in the simplest case is equal to $|x_i - y_i|$. For $\alpha = 2$ the vectors $||\mathbf{x}|| = 1$ are on the unit sphere, for large α the sphere is changed into a soft cuboid, for $\alpha = 1$ it becomes a pyramid and for $\alpha < 1$ it has hypocycloidal shape 3.

Many other distance functions may be used, such as the Mahalanobis distance:

$$D_M^2(\mathbf{x};\mathbf{t}) = \sum_{ij}(x_i - t_i)\Sigma^{-1}(x_i - t_i) \tag{15}$$

A more general quadratic distance function, with problem-specific, positive definite weight matrix Q, is:

$$D_Q(\mathbf{x},\mathbf{y};Q) = (\mathbf{x}-\mathbf{y})^{\mathrm{T}} Q(\mathbf{x}-\mathbf{y}) \tag{16}$$

where Q is a positive definite weight matrix. Various correlation factors are also suitable for metric functions, for example Camberra:

$$D_{Ca}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N} \frac{|x_i - y_i|}{|x_i + y_i|} \tag{17}$$

Chebychev:

$$D_{Ch}(\mathbf{x},\mathbf{y}) = \max_{i=1,\ldots,N} |x_i - y_i| \tag{18}$$

and Chi-square distance:

$$D_{\chi}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N} \frac{1}{\mathrm{sum}_i} \left(\frac{x_i}{\mathrm{size}_x} - \frac{y_i}{\mathrm{size}_y} \right)^2 \tag{19}$$

where sum_i is the sum of all values for attribute i occuring in the training set, and size_x and size_y are the sums of all values in vectors x and y.

The correlation distance measure is defined as:

$$D_{Cd}(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{N}(x_i - \bar{x_i})(y_i - \bar{y_i})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x_i})^2 \sum_{i=1}^{N}(y_i - \bar{y_i})^2}} \tag{20}$$

where $\bar{x_i}$ and $\bar{y_i}$ are the average values for attribute i occuring in the training set.

Kendall's Rank Correlation function is:

$$D_{KRC}(\mathbf{x},\mathbf{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{N} \sum_{j=1}^{i-1} \mathrm{sign}(x_i - x_j)\mathrm{sign}(y_i - y_j) \tag{21}$$

All these function are suitable to define radial components or to replace the Euclidean distance used in the definition of many transfer functions.

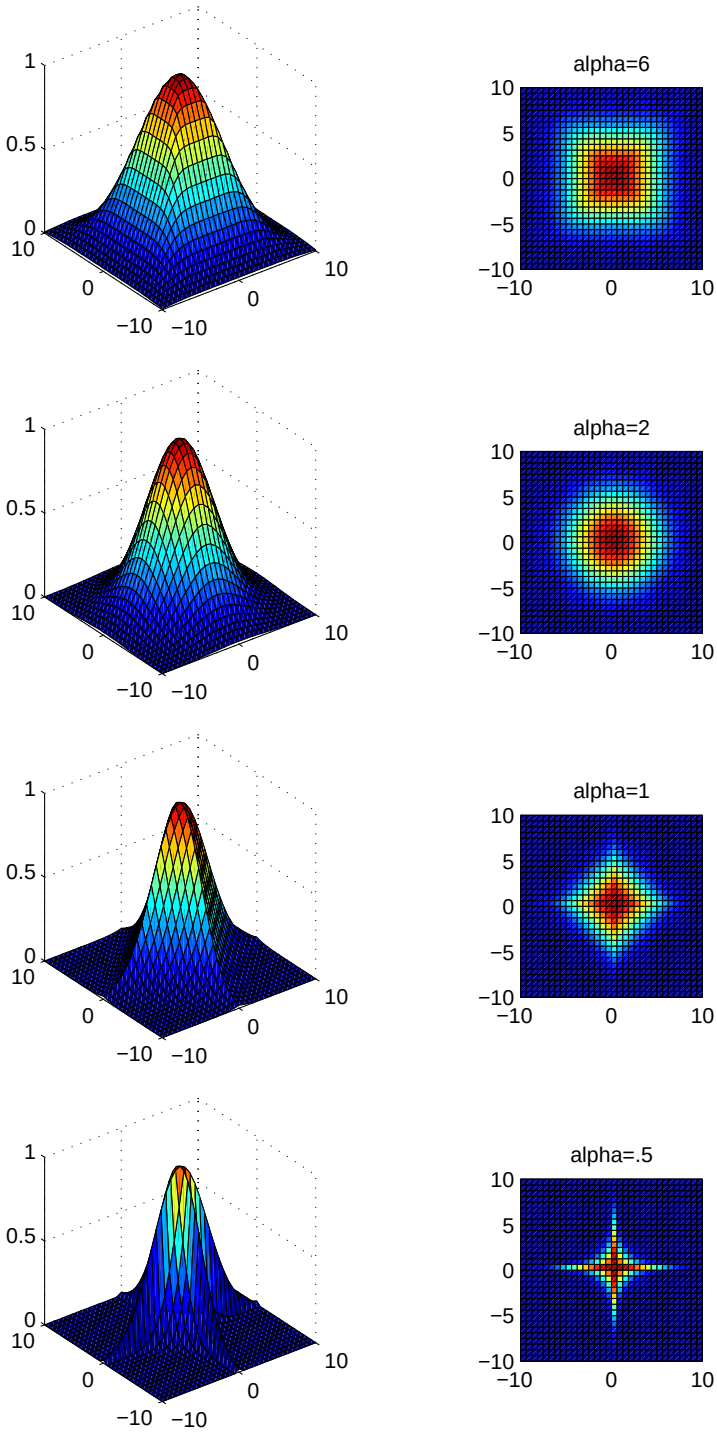

Figure 3: Gauss function with different Minkovski norms.

Heterogeneous Distance Functions. The distance function may be heterogeneous, using Minkovsky's metric for numerical features and probabilistic metric functions for symbolic features. In memory-based reasoning the Modified Value Difference Metric (MVDM) has gained popularity [44, 45, 46]. The distance between two N-dimensional vectors $\mathbf{x},\mathbf{y}$ with discrete (nominal, symbolic) elements, in a C class problem, is computed using conditional probabilities:

$$D_V^q(x,y) = \sum_{j=1}^{N}\sum_{i=1}^{C} \left| p(C_i|x_j) - p(C_i|y_j) \right|^q \tag{22}$$

where $p(C_i|x_j)$ is estimated by calculating the number $N_i(x_j)$ of times the value x_j of the feature j occurred in vectors belonging to class C_i, and dividing it by the number $N(x_j)$ of times x_j occurred for any class:

$$D_V^q(x,y) = \sum_{j=1}^{N}\sum_{i=1}^{C} \left| \frac{N_i(x_j)}{N(x_j)} - \frac{N_i(y_j)}{N(y_j)} \right|^q \tag{23}$$

A *value difference* for each feature j is defined as

$$d_V^q(x_j,y_j) = \sum_{i}^{C} \left| p(C_i|x_j) - p(C_i|y_j) \right|^q \tag{24}$$

Thus one may compute $D_V(\mathbf{x},\mathbf{y})$ as a sum of value differences over all features. Distance is defined here via a data-dependent matrix with the number of rows equal to the number of classes and the number of columns equal to the number of all attribute values. Generalization for continuous values requires a set of probability density functions $p_{ij}(x)$, with $i = 1,\ldots,C$ and $j = 1,\ldots,N$.

The Heterogeneous Euclidean-Overlap Metric (HEOM) is a simplified version of the VDM metric:

$$D_{HEOM}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{j=1}^{N} d_j(x_j,y_j)^2} \tag{25}$$

with attribute contributions equal to:

$$d_j(x_j,y_j) = \begin{cases} 1 & \text{if } x_j \text{ or } y_j \text{ is unknown} \\ overlap(x_j,y_j) & \text{if attribute } x_j \text{ is nominal} \\ \frac{|x_j - y_j|}{x_j^{max} - x_j^{min}} & \text{otherwise} \end{cases} \tag{26}$$

x_j^{max} and x_j^{min} are maximal and minimal values of j-th input attribute:

$$x_j^{max} = \max_i x_j^i \qquad x_j^{min} = \min_i x_j^i \tag{27}$$

The difference of x_j^{max} and x_j^{min} is the range of the j-th input variable (attribute). The $overlap$ is defined by:

$$overlap(x,y) \quad = \quad \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise} \end{cases} \tag{28}$$

The Heterogeneous Value Difference Metric (HVDM) is defined as:

$$D_{HVDM}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{j=1}^{N} \big(dh_j(x_j,y_j)\big)^2} \tag{29}$$

$$dh_j(x_j,y_j) = \begin{cases} 1 & \text{if } x_j \text{ or } y_j \text{ is unknown} \\ N\_vdm_j(x_j,y_j) & \text{if } x_j \text{ is nominal} \\ N\_dif_j(x_j,y_j) & \text{if } x_j \text{ is linear} \end{cases} \tag{30}$$

and

$$N\_dif_j(x_j,y_j) = \frac{|x_j - y_j|}{4\sigma_j} \tag{31}$$

where σ_j is the standard deviation of the numeric values of attribute x_j. Normalized VDM differences may be defined in several ways:

$$\begin{aligned} N1\_vdm(x,y) &= \sum_{i=1}^{C} \left| \frac{N_i(x)}{N(x)} - \frac{N_i(y)}{N(y)} \right| \\ N2\_vdm(x,y) &= \sqrt{\sum_{i=1}^{C} \left(\frac{N_i(x)}{N(x)} - \frac{N_i(y)}{N(y)} \right)^2} \\ N3\_vdm_j(x,y) &= \sqrt{C}\ N2\_vdm(x,y) \end{aligned} \tag{32}$$

The Discrete Value Difference Metric (DVDM) is used for continous inputs:

$$d_{DVDM}(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{N} vdm_j\big(disc_j(x_i), disc_j(y_j)\big)^2 \tag{33}$$

where $disc(x_j)$ is a discretization function:

$$disc_j(x_j) = \begin{cases} \left\lfloor \frac{x - x_j^{min}}{w_j} \right\rfloor + 1 & \text{if } x_j \text{ is continuous} \\ x_j & \text{if } x_j \text{ is discrete} \end{cases} \tag{34}$$

and w_j are parameters. Discretization allows application of VDM metrices to nominal as well as continuous inputs. Another way to compute VDM distances for continuous values is by using Interpolated Value Difference Metric:

$$d_{IVDM}(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{N} ivdm_j\big(x_j,y_j\big)^2 \tag{35}$$

where

$$ivdm_j(x_j,y_j) = \begin{cases} vdm_j(x_j,y_j) & \text{if } x_j \text{ is discrete} \\ \sum_{i=1}^{C} \big(p(C_i|x_j) - p(C_i|y_j)\big)^2 & \text{otherwise} \end{cases} \tag{36}$$

Probabilities appearing in the definition given above are calculated by interpolation:

$$p(C_i|x_j) = P(C_i|x_j,u) + \frac{x_j - x^{mid}_{j,u}}{x^{mid}_{j,u+1} - x^{mid}_{j,u}}(P(C_i|x_j,u+1) - P(C_i|x_j,u)) \tag{37}$$

where $x^{mid}_{j,u}$ and $x^{mid}_{j,u+1}$ are midpoints of two consecutive discretized ranges such that $x^{mid}_{j,u} \leq x_j \leq x^{mid}_{j,u+1}$, $P_{j,u,c}$ is the probability value of the discretized range u, defined at the midpoint of range u, and values of u are found by first setting $u = disc_j(x_j)$.

Using VDM-type metrics leads to problems with calculation of gradients. Purely numerical input vectors are obtained using continuous feature values and replacing symbolic and discrete attributes with $p(C_i|x_j)$ probabilities. Resulting numerical vectors have the number of components equal to the number of different symbolic values times the number of classes. Distances calculated with such input vectors are identical to those obtained with the heterogenous distance functions.

3.2 Combination of inner product and distance based activation

To represent complex decision borders correctly, transfer functions may need sophisticated activation functions. A good example of such activation has been provided by Ridella *et al*. [47]:

$$A_R(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{N} w_i x_i + w_{N+1}\sum_{i=1}^{N} x_i^2 \tag{38}$$

Another mixed activation is used in *conical* transfer function described by Dorffner [13]:

$$A_C(\mathbf{x};\mathbf{w},\mathbf{t},\omega) = I(\mathbf{x}-\mathbf{t};\mathbf{w}) + \omega D(\mathbf{x}-\mathbf{t}) \tag{39}$$

Transfer functions C_{GL1} and C_{GL2} (Eq. 88 and 89) use another form of mixed activations:

$$A_{GL1} = \mathbf{w}^T\mathbf{x} + \alpha||\mathbf{x}-\mathbf{t}|| \tag{40}$$

$$A_{GL2} = \alpha(\mathbf{w}^T\mathbf{x})^2 + \beta||\mathbf{x}-\mathbf{t}||^2 \tag{41}$$

These activations are of the scalar type. Bicentral transfer functions (described in detail in section 5.6) use vector type activations. Furthermore, bicentral functions use two vectors of activations, left and right $A_i = \{A_i^+, A_i^-\}$, and $\mathbf{A} = [A_1, \ldots, A_n]$. Below different bicentral activations are presented:

$$\text{Bi} \quad A1_i^{\pm} = s_i(x_i - t_i \pm b_i), \tag{42}$$

$$\text{Bi2s} \quad A2_i^{\pm} = s_i^{\pm}(x_i - t_i \pm b_i), \tag{43}$$

$$\text{BiR} \quad A3_i^{\pm} = s_i(x_i + r_i x_{i+1} - t_i \pm b_i), \tag{44}$$

$$\text{BiR2s} \quad A4_i^{\pm} = s_i^{\pm}(x_i + r_i x_{i+1} - t_i \pm b_i) \tag{45}$$

The usefulness of such activation functions will become clear in section 5.6.

4 OUTPUT FUNCTIONS

In the simplest case the identity function may be used for the output function. This is done in linear networks or if the radial coordinate function $||\mathbf{x}-\mathbf{t}||$ is used as an output function – it may be treated as a distance-based activation. Since activation functions are in most cases unbounded, output functions are used to limit the signals propagated through the network. There are three major choices here:

- Sigmoidal non-local functions.

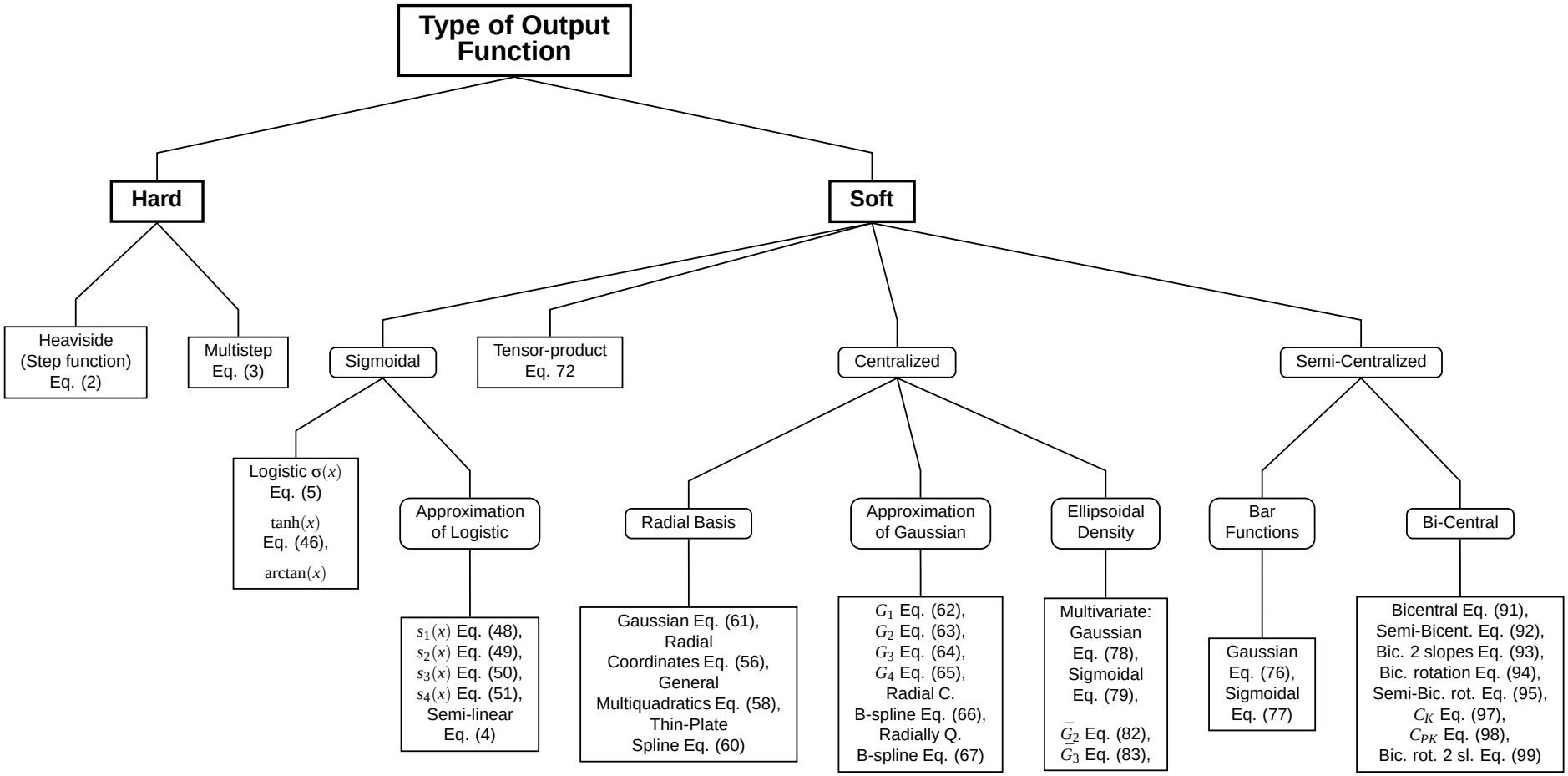

Figure 4: Taxonomy of output functions.

- Functions localized around a single center.
- Semi-centralized functions that have either many centers or hard to define centers.

Please note that localization properties are true only for output functions, for example treating $\sigma(x)$ as a function of a scalar variable x. Non-local functions may be combined with localized activation functions giving localized behavior of the total transfer functions.

4.1 Sigmoidal output functions

Sigmoidal output functions (*sigmoid* means S-shaped) are not only natural from the statistical point of view but are also good squashing functions for unbounded activation. Sigmoidal output functions have non-local behavior, i.e. for large activations they are non-zero in an infinite domain. Output functions may also be localized around some value.

Sigmoidal functions are smooth and – what is very important for backpropagation algorithm – it is easy to calculate their derivatives. For the logistic function Eq. (5) the derivative is $\sigma(I)' = \sigma(I)(1-\sigma(I))$. Logistic functions may be replaced by the error (erf) function, arcus tangent or the hyperbolic tangent functions:

$$\tanh(I/s) = \frac{1-e^{-I/s}}{1+e^{-I/s}} \tag{46}$$

$$\tanh'(I/s) = \mathrm{sech}^2(I/s)/s = \frac{4}{s(e^{-I/s}+e^{+I/s})^2} = (1-\tanh(I/s)^2)/s \tag{47}$$

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

$$s_1(I;s) = \Theta(I)\frac{I}{I+s} - \Theta(-I)\frac{I}{I-s} = I\frac{\mathrm{sgn}(I)I-s}{I^2-s^2} \tag{48}$$

$$s_2(I;s) = \frac{sI}{1+\sqrt{1+s^2I^2}} = \frac{sI}{1+q} \tag{49}$$

$$s_3(I;s) = \frac{sI}{1+|sI|} \tag{50}$$

$$s_4(I;s) = \frac{sI}{\sqrt{1+s^2I^2}} \tag{51}$$

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

$$s_1'(I;s) = \frac{s}{(I+s)^2}\Theta(I) + \frac{s}{(I-s)^2}\Theta(-I) = \frac{s}{(I+\mathrm{sgn}(I)s)^2} \tag{52}$$

$$s_2'(I;s) = \frac{s}{q(1+q)} \tag{53}$$

$$s_3'(I;s) = -\mathrm{sgn}(I)\frac{s^2I}{(1+|sI|)^2} + \frac{s}{1+|sI|} \tag{54}$$

$$s_4'(I;s) = -\frac{s^3I^2}{(1+x^2)^{3/2}} + \frac{s}{\sqrt{1+x^2}} \tag{55}$$

Shapes of these functions[1] are compared in Fig. 5. 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

[1] All functions were linearly transformed to obtain output between 0 and 1; their slope parameters s are chosen to make them as similar to each other as possible.

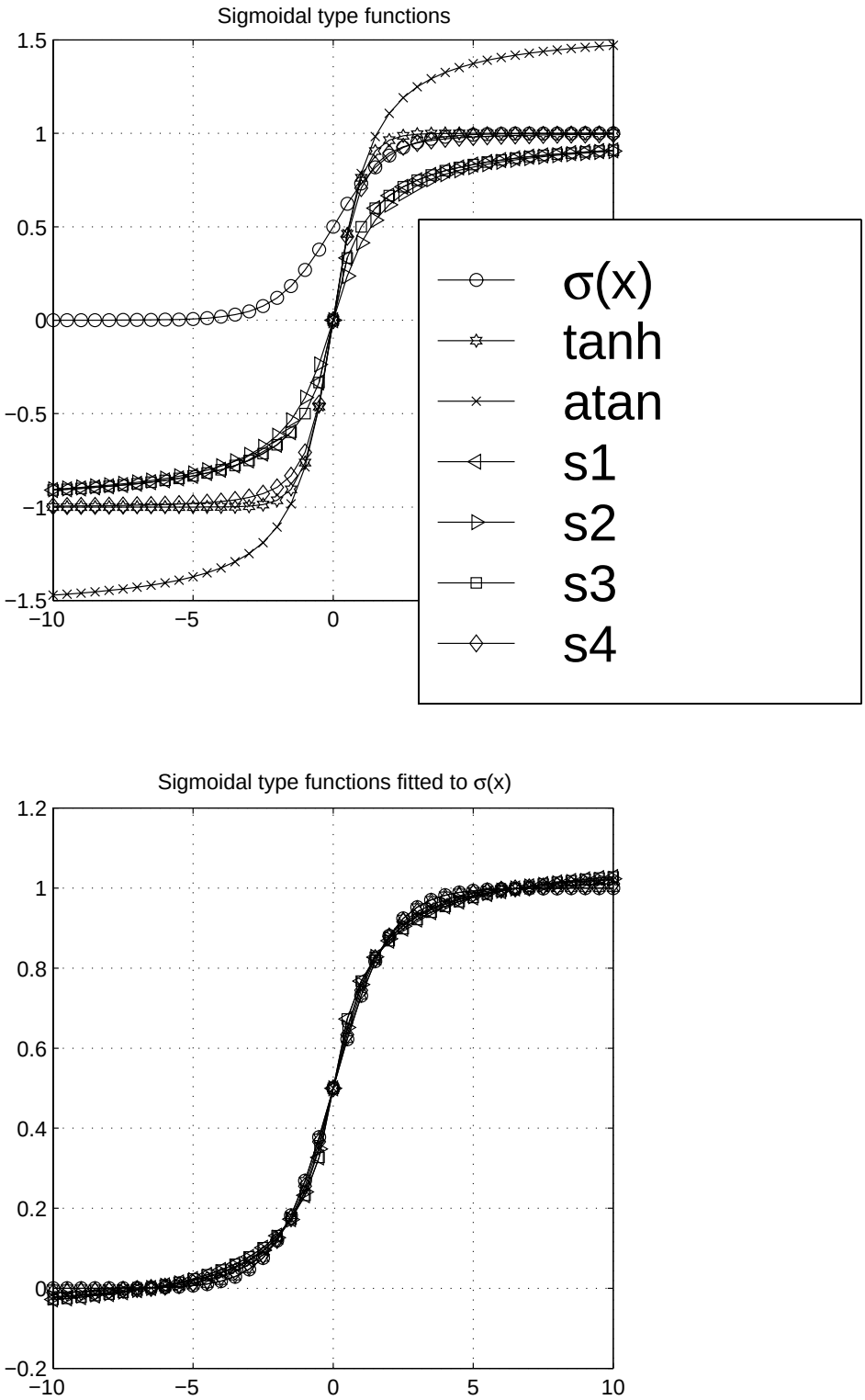

Figure 5: Comparison of sigmoidal transfer functions.

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. Another approximation speeding up calculations of exponents is presented in [48].

For sigmoidal functions powerful mathematical results exist, showing that a universal approximator may be built from neural networks with single hidden layer of processing elements [6, 7]. This is not surprising, since almost any set of functions may be used as a basis for universal approximator. What is more interesting are the estimates of the rates of convergence using sigmoidal functions. For a single layer network of n units under some weak assumptions about the approximated function the rate of convergence improves as $O(n^{-\frac{1}{2}})$, i.e. it does not depend on the dimension of the problem [49, 50, 51]. For polynomial functions the rate depends on the dimension d and is $O(n^{-\frac{1}{2d}})$ which for multidimensional problems is unacceptably slow. For that reason we are quite sceptical about the use of orthogonal polynomials as transfer functions [52, 53] for high dimensional problems. Other non-polynomial functions, such as periodic functions, wavelets and some localized functions share with sigmoidal functions independence of convergence rates from the dimensionality of the problem [54].

4.2 Functions localized around a single center

Another class of powerful functions used in approximation theory [4, 55, 56] is called the radial basis functions (RBFs). Except for approximation theory these types of functions have also been in use for many years in pattern recognition under different names (cf. potential function approach, [57]). A very good introduction to RBF and more general regularization networks was given by Poggio and Girosi [58] (see also [1, 9, 13, 59, 60, 61, 62, 63]).

Radial basis functions take the radial coordinate $r = ||\mathbf{x}-\mathbf{t}||$ for an activation. In this section we are only interested in the output functions $o(r) = o(I)$ used in this context. Some of these output functions are non-local, while others are local. The nonlocal radial coordinate function (see Fig. 6) is the simplest:

$$h(r) = r = ||\mathbf{x}-\mathbf{t}|| \tag{56}$$

For approximation problems Allison [64] recommends simple multiquadratic functions:

$$s_m(r;b) = \sqrt{b^2+r^2}; \qquad s'_m(r;b) = \frac{r}{s_m(r;b)} \tag{57}$$

where b is the smoothness parameter. Other examples of RBFs include the nonlocal general multiquadratics, and thin-plate spline functions (see Fig. 7, 8):

$$h_1(r,b) = (b^2+r^2)^{-\alpha}, \quad \alpha > 0 \tag{58}$$

$$h_2(r,b) = (b^2+r^2)^{\beta}, \quad 0 < \beta < 1 \tag{59}$$

$$h_3(r,b) = (br)^2 \ln(br) \tag{60}$$

Several types of localized radial basis functions exist. Among them *Gaussian functions* (see Fig. 9) are unique since for Euclidean distance functions (and other distance functions that may be presented as a sum of independent components) they are separable (see [65] on the importance of separability). Separable functions are expressed as products of independent factors for each of the input components, i.e. $f(\mathbf{x}) = \prod f_i(x_i)$.

$$G(r,b) = e^{-r^2/b^2} \tag{61}$$

Although the processing power of neural networks based on non-local processing units does not depend strongly on the type of non-polynomial neuron processing functions such is not the case for localized units. Gaussian functions e^{-r^2} are quite simple but not the least expensive to compute. Logistic function, tanh or simple quadratic and quartic functions with localized activation approximate roughly the shape of a Gaussian function:

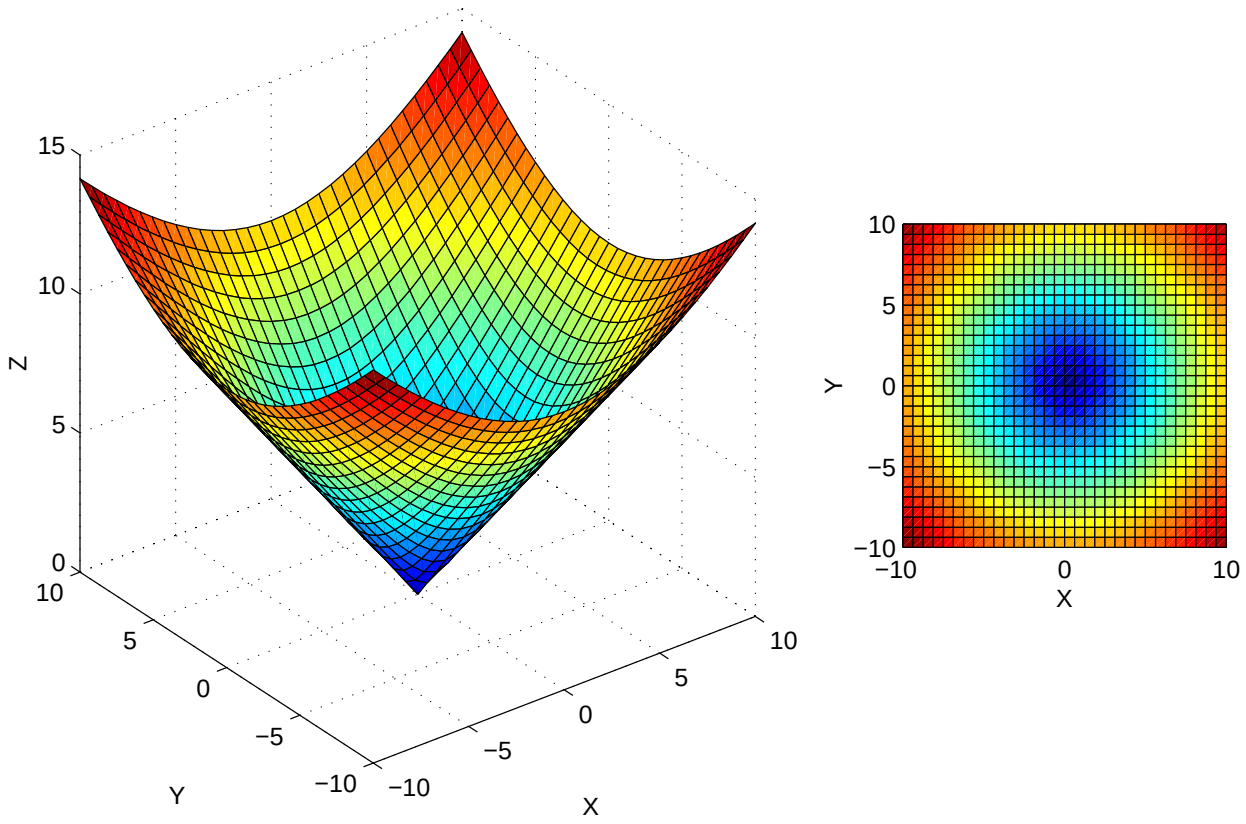

Figure 6: Radial Coordinates function (Eq. 56).

$$G_1(r) = 2 - 2\sigma(r^2) \tag{62}$$

$$G_2(r) = 1 - \tanh(r^2) \tag{63}$$

$$G_3(r) = \frac{1}{1+r^2}; \quad G_3'(r) = -2rG_3^2(r); \tag{64}$$

$$G_4(r) = \frac{1}{1+r^4}; \quad G_4'(r) = -4r^3G_4^2(r) \tag{65}$$

Radial cubic B-spline function were used in [66]. They are defined by:

$$RCBSpline(r) = \frac{1}{10h^3}\begin{cases} h^3 + 3h^2(h-r) + 3h(h-r)^2 + 3(h-r)^3 & r \le h \\ (2h-r)^3 & h < r \le 2h \\ 0 & 2h < r \end{cases} \tag{66}$$

where $r = ||\mathbf{x} - \mathbf{t}||^2$ and $\mathbf{t}$ is a center. Fig. 10) shows an example of such function.

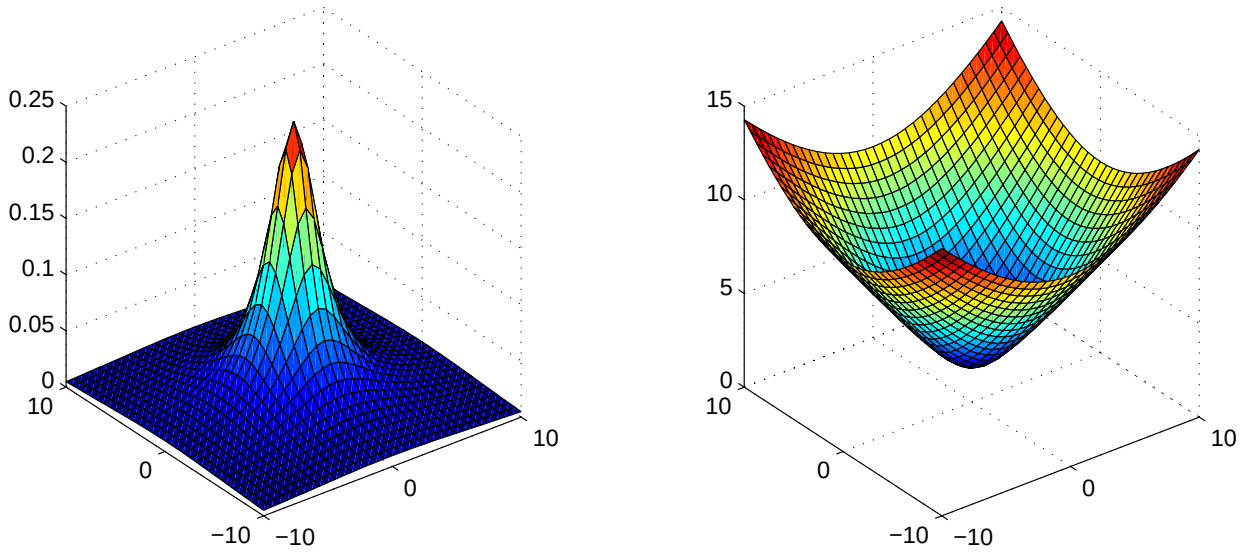

Figure 7: General Multiquadratics function (Eq. 58).

Radially quadratic B-spline function were also used in [66] and are defined by:

$$RQBSpline(r) = \frac{1}{3h^2}\begin{cases} -2r^2+3h^2 & r \le h \\ (2h-r)^2 & h < r \le 2h \\ 0 & 2h < r \end{cases} \tag{67}$$

where $r = ||\mathbf{x} - \mathbf{t}_i||$ and $\mathbf{t}_i$ is a center (see Fig. 11; please note that there are errors in definition of these functions in [66]). In this case output functions are a bit more complicated than in most other cases, but the shapes of these functions are similar to Gaussians. Changing the activation to a non-Euclidean distance function has strong influence on the contours of these functions.

Comparison of all Gaussian-like functions is presented in Fig. 12.

Networks based on radial functions are also universal approximators [8, 9]. 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 [67].

The rate of convergence of the Radial Basis Function networks for fixed dispersions has been determined by Niyogi and Girosi [68]. Since the true function is unknown an error may only be measured in respect to the best possible (Bayes) estimate of this function, called the regression function $f_0(X)$. The distance between the regression function and the function realized by the radial basis functions network with n nodes, each of d dimensions, given k examples, estimated with the confidence (probability) $1-\delta$, is:

$$E\left[(f_0(X) - F_{n,k}(X))^2\right] = \int_X dX P(X)\,(f_0(X) - F_{n,k}(X))^2 \le O\left(\frac{1}{n}\right) + O\left(\sqrt{\frac{nd\ln(nk) - \ln\delta}{k}}\right) \tag{68}$$

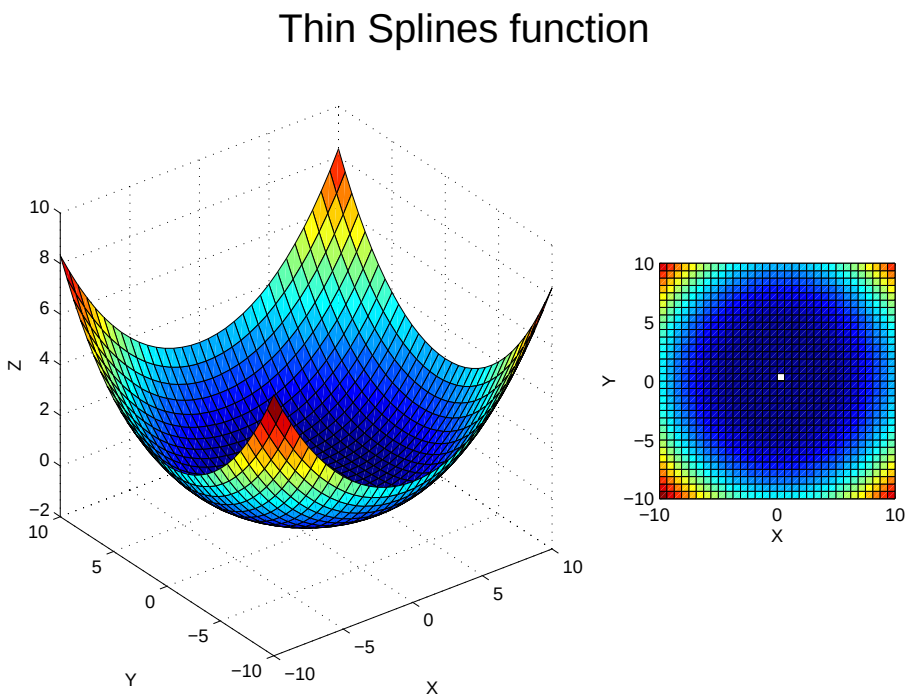

Figure 8: Thin-plate spline function (Eq. 60).

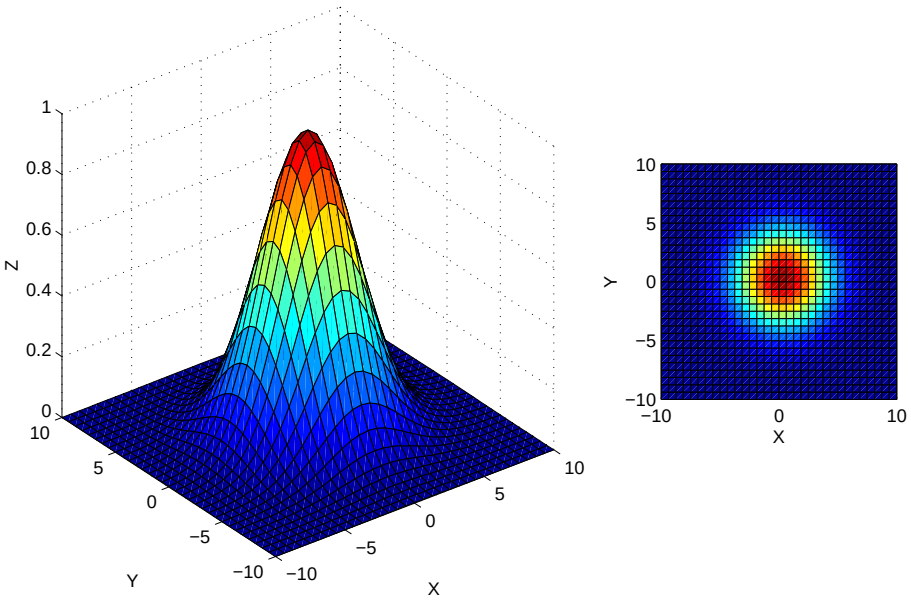

Figure 9: Gaussian function (Eq. 61).

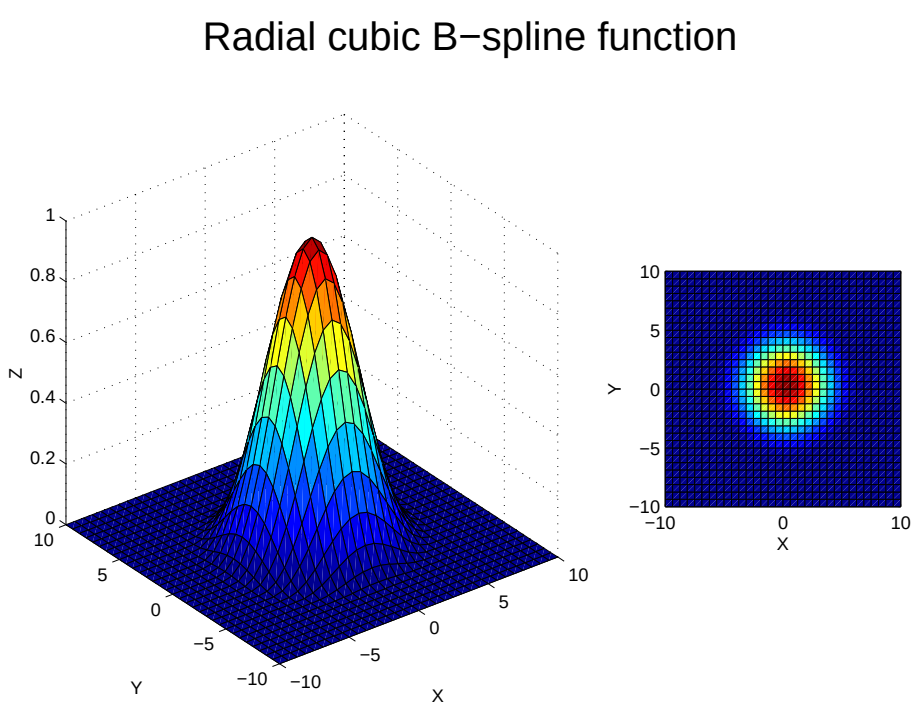

Figure 10: Radial cubic B-spline function (Eq. 66).

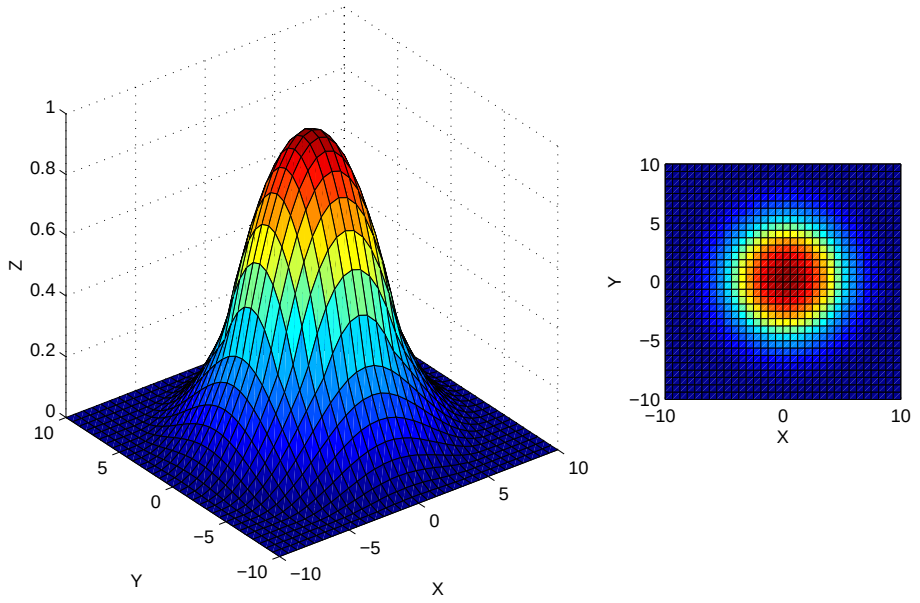

Figure 11: Radially quadratic B-spline function (Eq. 67).

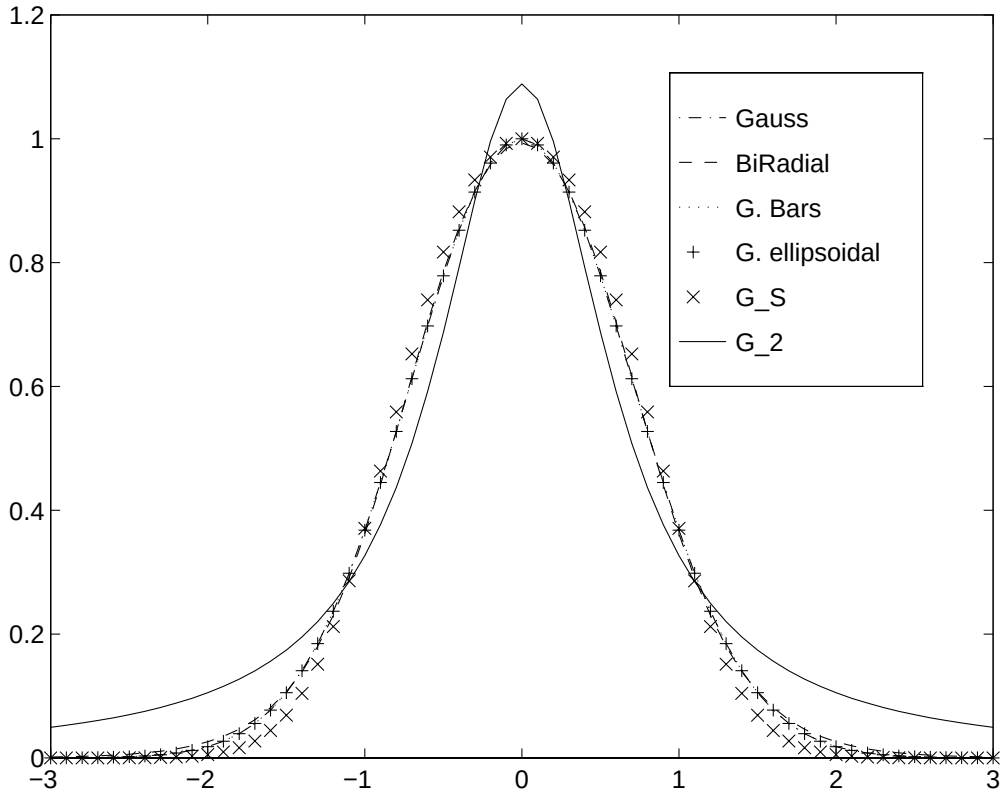

Figure 12: Comparison of Gaussian-like functions Eq. 61, 91, 62–66.

Approximation theory determines the first factor, $O(1/n)$, while statistics the second factor. The error vanishes only if the network complexity, expressed by the number of its nodes n, tends to infinity slower than the number of data samples k. For any fixed number of data points there is an optimal number of network nodes that minimizes the generalization error.

4.3 Semi-centralized functions

Semi-centralized output functions operate on vector components of activation. For example, the Gaussian bar functions:

$$\bar{G}(\mathbf{r},\mathbf{b},\mathbf{v}) = \sum_{i=1}^{N} v_i e^{-r_i^2/b_i^2}; \quad r_i = (x_i - t_i) \tag{69}$$

or the bicentral functions:

$$Bi(\mathbf{r};\mathbf{b},\mathbf{s}) = \prod_{i=1}^{N} \sigma(I^+)(1-\sigma(I^-)) \tag{70}$$

where I^+ and I^- are defined by Eq. (42–45) These functions will be discussed in details below.

5 TRANSFER FUNCTIONS

In this section activation and output functions are combined together to give the final transfer function. We have divided transfer functions into those that are nonlocal, those that are local or semilocal, with separate subsection on functions with hiperellipsoidal contours, and those that may become local or nonlocal depending on their adaptive parameters, called for simplicity *universal functions*. The last subsection deals with universal bicentral functions, since several variants of these functions are described.

5.1 Nonlocal 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. Typical transfer functions used in multilayer perceptrons (MLPs) for discrimination and approximation are sigmoidal (Eq. 5,46 – 48) with the fan-in activation (1).

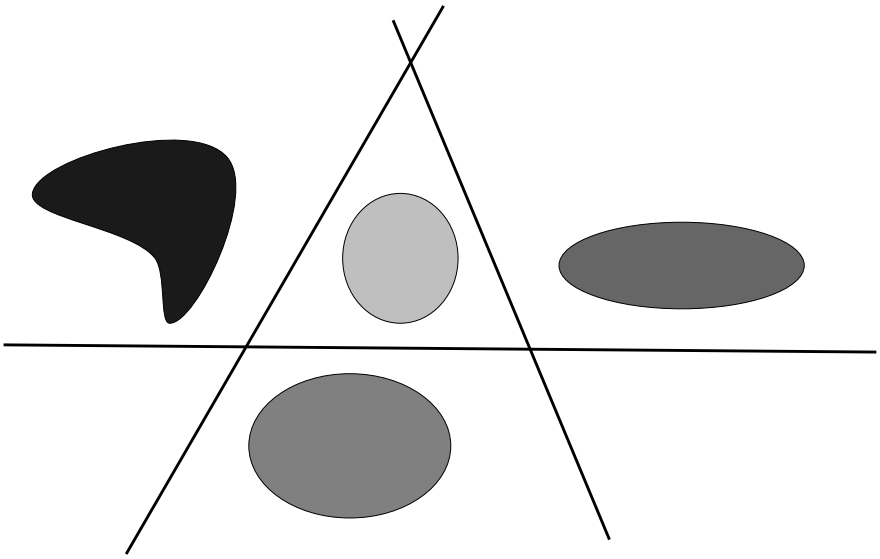

Figure 13: Decision regions formed using sigmoidal transfer functions.

The classification decision regions of neural networks based on such transfer functions are formed by cutting the input space with hyperplanes (Fig. 13) and thus are suitable for calculation of posterior probabilities. The system *pretends* that it knows everything – this may be quite improper, especially far from the training 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 may be desirable, but for general mappings it limits the precision of the adaptive system.

Radial Basis Functions are used as transfer functions in many neural network simulators, but in most simulators only Gaussian functions are provided. Nonlocal radial transfer functions include general multiquadratics (Eq. 58), and thin-plate spline functions (Eq. 60, Fig. 7, 8), which may be combined with Minkovsky or many other distance functions.

The *Lorentzian* response functions (see Fig. 14) introduced by Giraud *et al.* [14] used the squared fan-in activation function to create surfaces of constant density of non-localized window-type:

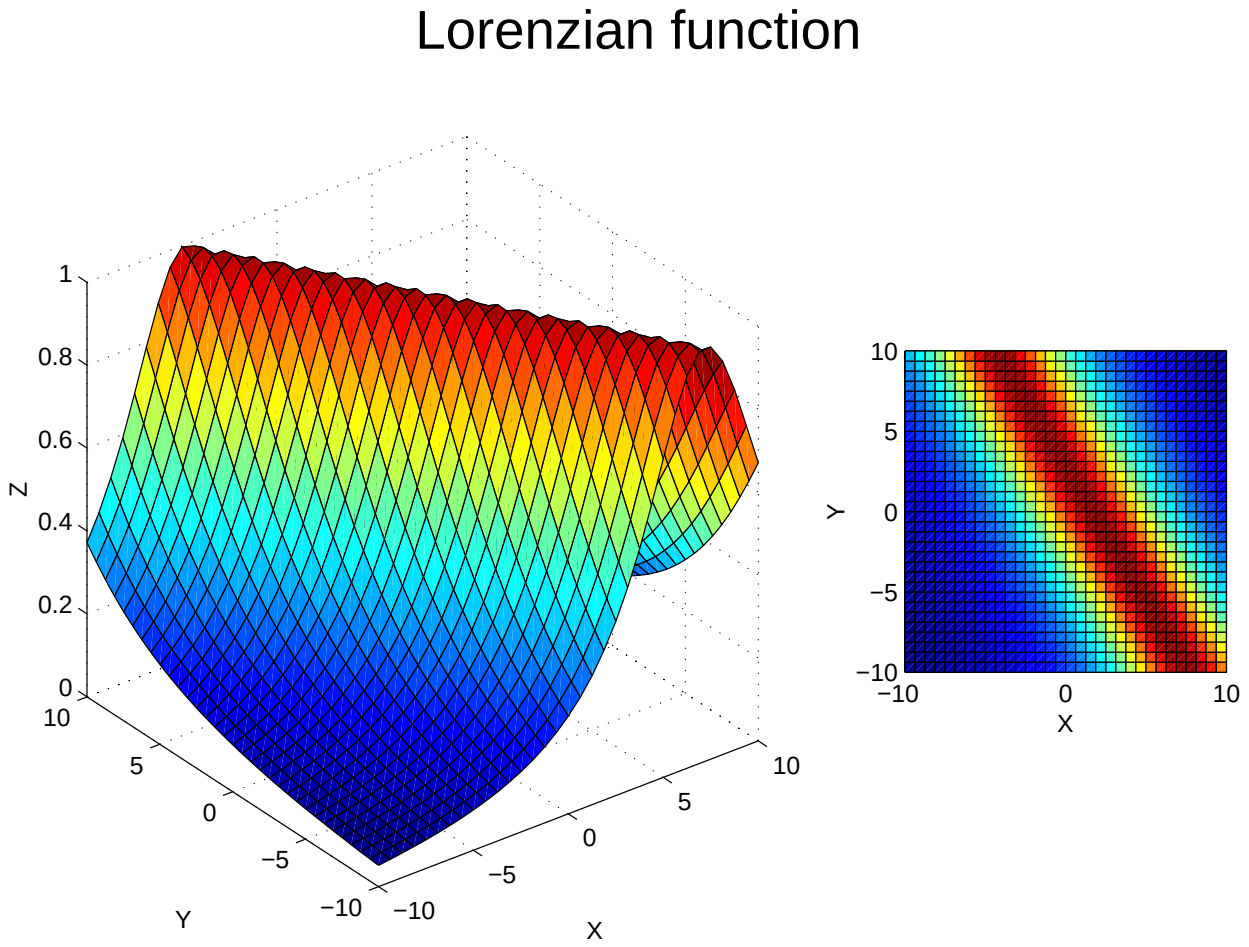

Figure 14: Lorenzian function (Eq. 71).

$$L(\mathbf{x};\mathbf{w}) = \frac{1}{1+I^2(\mathbf{x};\mathbf{w}} = \frac{1}{1+\left(\sum_{i=1}^{N} w_i x_i - \theta\right)^2} \tag{71}$$

with the half-width equal to $1/\sqrt{\sum_i w_i^2}$. Non-local functions of window type may also be obtained from many separable local and semi-local transfer functions, described below, if the product of individual components of these functions does not cover all dimensions.

The tensor-product truncated power basis was used in MARS algorithm by Friedman [69] (see Fig. 15)

$$T(\mathbf{x};\mathbf{t},\mathbf{s},I) = \prod_{i\in I}[s_i(x_i - t_i)]_+^q \tag{72}$$

where I is a subset of input features, q is a user-defined variable, $\mathbf{t}$ is the center, s_i defines direction and is equal to 1 or -1, and $[\,\cdot\,]_+$ indicates positive support. Such tensor product functions are used in the MARS algorithm to build an

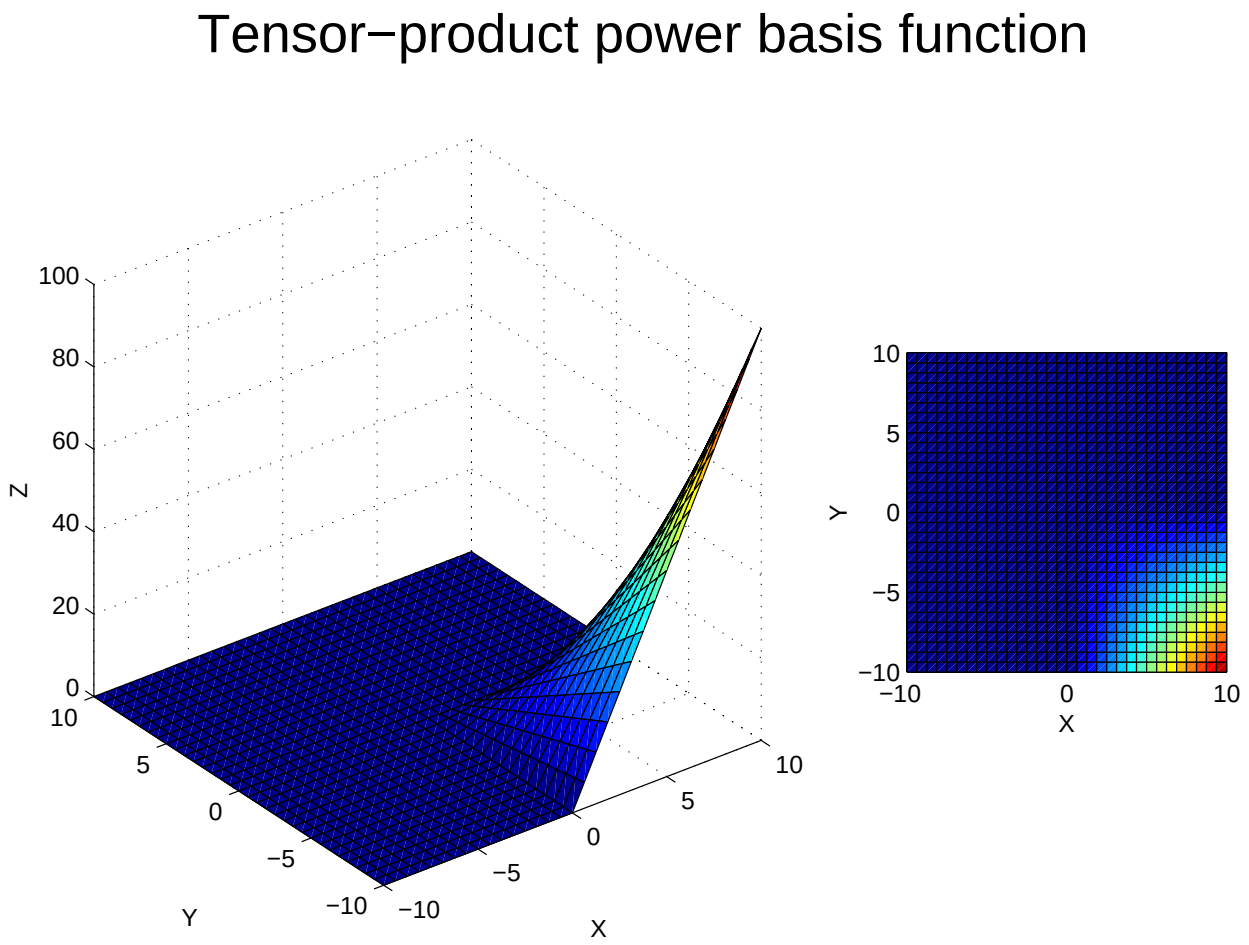

Figure 15: Tensor-product power basis function.

approximating function as a linear combination:

$$MARS(\mathbf{x};\mathbf{w},\mathbf{T},\mathbf{S},I) = \sum_{i=1}^{N} w_i\, T(\mathbf{x};\mathbf{t}_i,\mathbf{s}_i,I_i) + w_0 \tag{73}$$

5.2 Local and Semi-local Transfer Functions

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. Early systems based on such functions may be traced back to the older work on pattern recognition [57]. Moody and Darken [70] 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 [71] have shown the power of local training algorithms in a more general way. According to Kadirkamanathan and Niranjan [72] smoothness conditions for adding new units in constructive neural networks are satisfied only by strongly local units.

Radial Basis Functions are used as transfer functions in many neural network simulators, with Gaussian functions being the most popular. The activation function is usually taken as the Euclidean distance, but it is easily generalized

to an arbitrary distance function $D(\mathbf{x};\mathbf{t})$, where $\mathbf{t}$ is the center of the unit, an adaptive parameter, such that the activation has minimum for $\mathbf{x}=\mathbf{t}$ and grows in an unbounded way if $\mathbf{x}$ is far from $\mathbf{t}$. Hamming distance is frequently used for binary inputs. Additional adaptive parameters may be introduced as scaling factors (cf. Eq. 14) in each dimension (N parameters), or as one common scaling factor for each center.

The simplest approach, used in the RBF networks, is to set a number of radial functions $G_i(\mathbf{x})$ with predetermined parameters b_i and positions $\mathbf{t}_i$ (for example, positions are set by k-means clustering and dispersions to twice the nearest neighbor distance [73]) and determine the linear coefficients w_i in the approximation function:

$$f(\mathbf{x},\mathbf{w},\mathbf{b},\mathbf{t}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x},b_i,\mathbf{t}_i) = \sum_{i=1}^{M} w_i e^{-||\mathbf{x}-\mathbf{t}_i||^2/b_i^2} \tag{74}$$

In regularization networks the centers $\mathbf{t}_i$ of each of the radial units are also optimized [58], allowing for reduction of the number of basis functions in the presence of noisy data (such reduction corresponds 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, for each neural unit.

Moody and Darken used a normalized version of the Gaussian function:

$$GN_i(\mathbf{x};\mathbf{t}_i,b_i) = \frac{e^{-||\mathbf{x}-\mathbf{t}_i||^2/b_i^2}}{\sum_{j=1}^{M} e^{-||\mathbf{x}-\mathbf{t}_j||^2/b_j^2}} \tag{75}$$

An interesting property of this function is that for a given $\mathbf{x}$, the sum over all GN_i functions is equal to 1. Therefore the output may be treated as the probability computed by neuron i. In Fig. 16 output from 4 normalized Gaussian functions is presented. Bridle [74] calls such normalized functions *soft-max*.

5.3 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 [8] and by Park and Sandberg [9]. Instead of multidimensional Gaussian functions these authors advocate a combination of one-dimensional Gaussians (see Fig. 17):

$$\bar{G}(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{v}) = \sum_{i=1}^{N} v_i e^{-(x_i-t_i)^2/b_i^2} \tag{76}$$

In this case separation of the activation and the output functions is not so natural. $3N$ 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 involve 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 functions 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 [70]). 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 *sigmoidal bar function* (see Fig. 18):

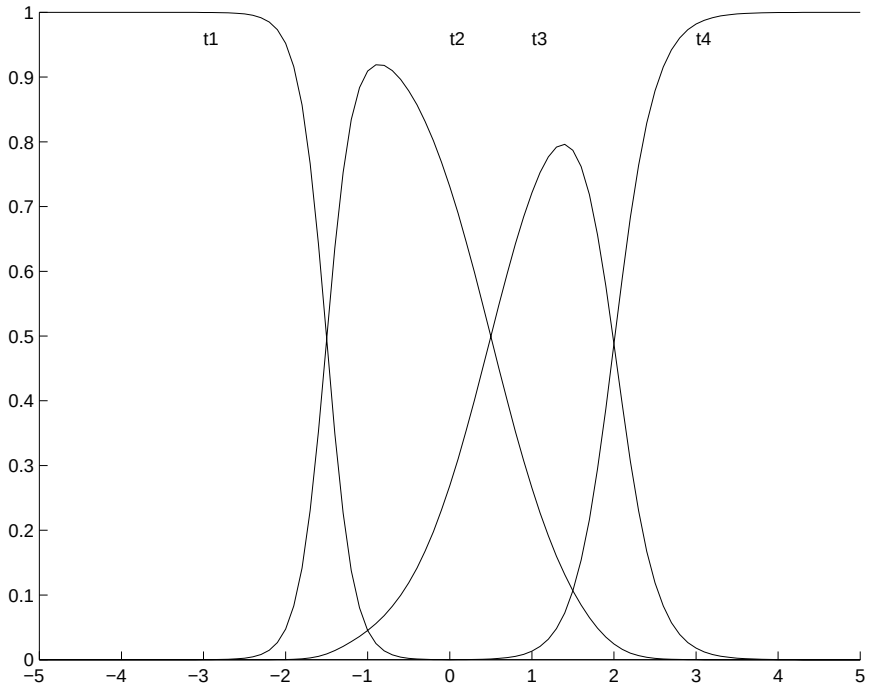

Figure 16: Normalized Gaussian function called also softmax.

$$\bar{\sigma}(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{v}) = \sum_{i=1}^{N} \frac{v_i}{1+e^{-(x_i-t_i)^2/b_i^2}} \tag{77}$$

These functions, similarly to Gaussian bars, give contours of constant densities that cannot be rotated easily, which is clearly a disadvantage. Sigmoidal bar functions should not be used to represent data clustered only around a few points, 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 $2k$ sigmoidal or Gaussian *bars* in the input space are sufficient to represent such data and $2k-2$ hyperplanes or sigmoids are sufficient for discrimination of each cluster.

5.4 Functions with ellipsoidal contours

The multivariate Gaussian functions give localized hyperellipsoidal output densities (see Fig. 19):

$$G_g(\mathbf{x};\mathbf{t},\mathbf{b}) = e^{-D^2(\mathbf{x};\mathbf{t},\mathbf{b})} = \prod_{i=1}^{N} e^{-(x_i-t_i)^2/b_i^2} \tag{78}$$

Dispersions b_i may be interpreted as scaling factors in the Euclidean distance function. A similar result is obtained by combining the sigmoidal output function (or any other logistic function) with the quadratic distance function (see Fig. 20), for example:

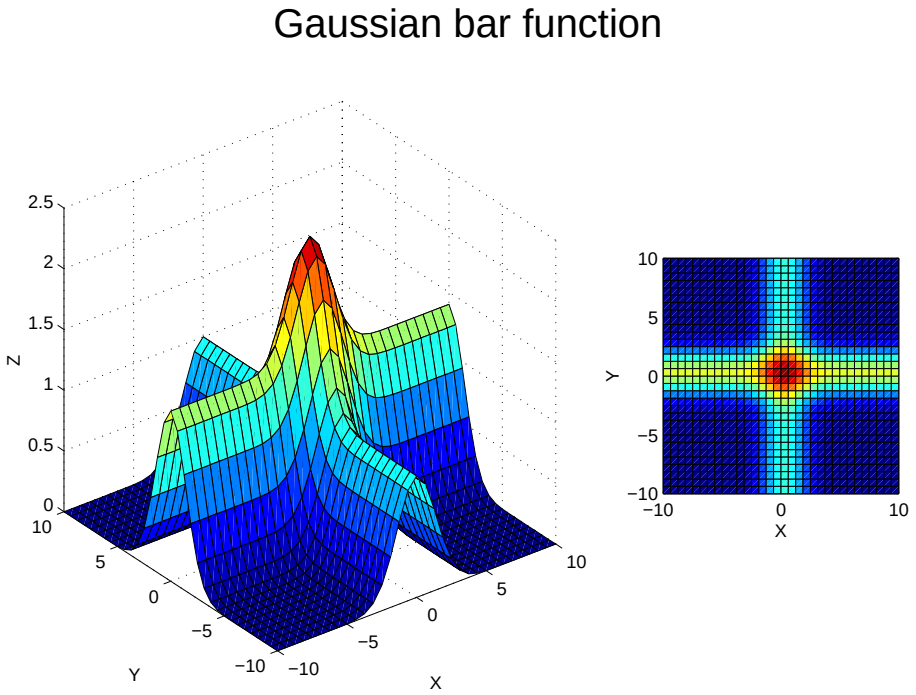

Figure 17: Gaussian bar function (Eq. 76).

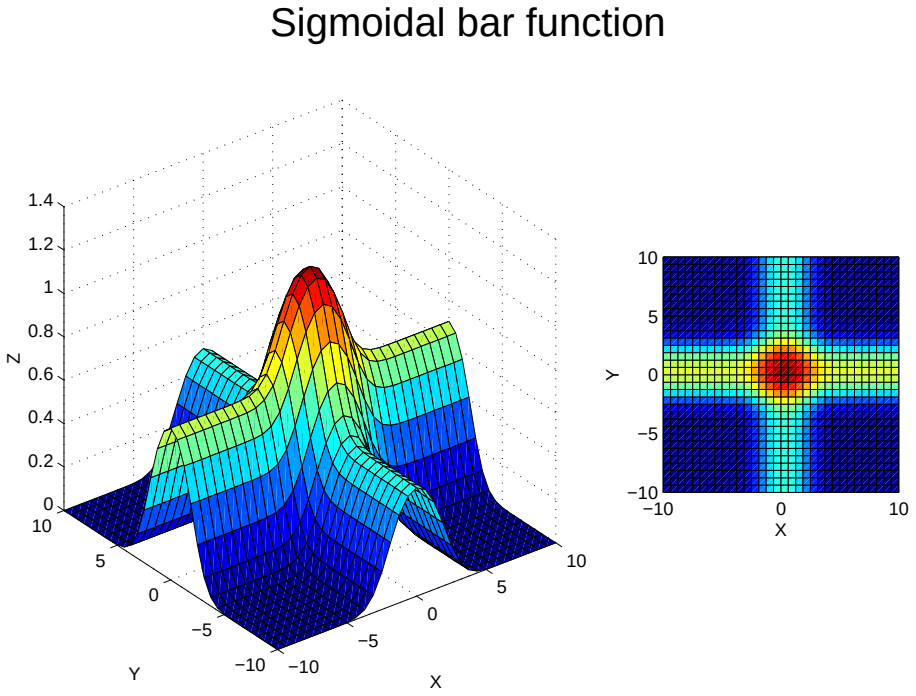

Figure 18: Sigmoidal bar function (Eq. 77).

$$G_S(\mathbf{x};\mathbf{t},\mathbf{b}) = 1-\sigma(D(\mathbf{x};\mathbf{t},\mathbf{b})^2) \quad (79)$$

$$= \frac{1}{1+\prod_{i=1}^{N} e^{(x_i-t_i)^2/b_i^2}} = \frac{1}{1+e^{D^2(\mathbf{x};\mathbf{t},\mathbf{b})}} \quad (80)$$

For N-dimensional input space each ellipsoidal unit uses $2N$ adaptive parameters. Taking the Mahalanobis distance function $D_M(\mathbf{x};\mathbf{t})$, Eq. (15), with (symmetric) covariance matrix Σ, 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:

$$D_Q^2(\mathbf{x};\mathbf{Q};\mathbf{t}) = \sum_{i\geq j} Q_{ij}(x_i-t_i)(x_j-t_j) \quad (81)$$

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 using too many nonlinear parameters per neural node. Simpler units giving approximately ellipsoidal densities are also useful, for example (see Fig. 21):

$$\bar{G}_2(\mathbf{x};\mathbf{t},\mathbf{b}) = \prod_{i=1}^{N} \frac{1}{1+(x_i-t_i)^2/b_i^2} \quad (82)$$

This formula can not 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 (see Fig. 22):

$$\bar{G}_3(\mathbf{x};\mathbf{t},\mathbf{b}) = \frac{1}{1+\sum_{i=1}^{N}(x_i-t_i)^2/b_i^2} = \frac{1}{1+D^2(\mathbf{x};\mathbf{t},\mathbf{b})} \quad (83)$$

These functions also give hyperellipsoidal densities. 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. 78) is that after a simple renormalization:

$$G_R(\mathbf{x};\mathbf{t},\mathbf{b}) = \frac{G_g(\mathbf{x};\mathbf{t},\mathbf{b})}{G_g(\mathbf{x};\mathbf{t},\mathbf{b})+G_g(\mathbf{x};-\mathbf{t},\mathbf{b})} = \frac{1}{1+e^{-4\sum_{i=1}^{N} x_i t_i/b_i^2}} = \sigma(\mathbf{w}\cdot\mathbf{x}) \quad (84)$$

they become non-local and are equivalent to sigmoidal functions $\sigma(\mathbf{w}\cdot\mathbf{x})$, where $w_i = 4t_i/b_i^2$. In this way RBF networks may be used instead of MLP networks and vice versa, a simple input transformation allows MLP networks to be used as RBF networks with localized transfer functions [75, 76].

5.5 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 are localized, and for other parameters non-localized. Several functions of this kind have been proposed recently. Ridella *et al*. [47] use circular units in their Circular Backpropagation Networks. The output function is a standard sigmoid while the activation function contains one extra term (see Fig. 23):

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

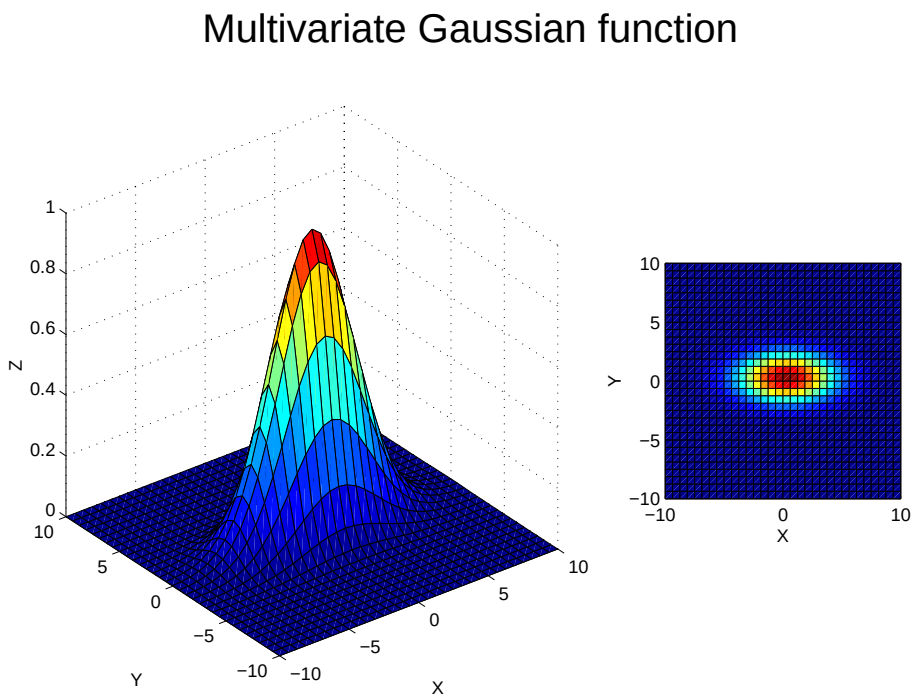

Figure 19: Multivariate Gaussian function (Eq. 78).

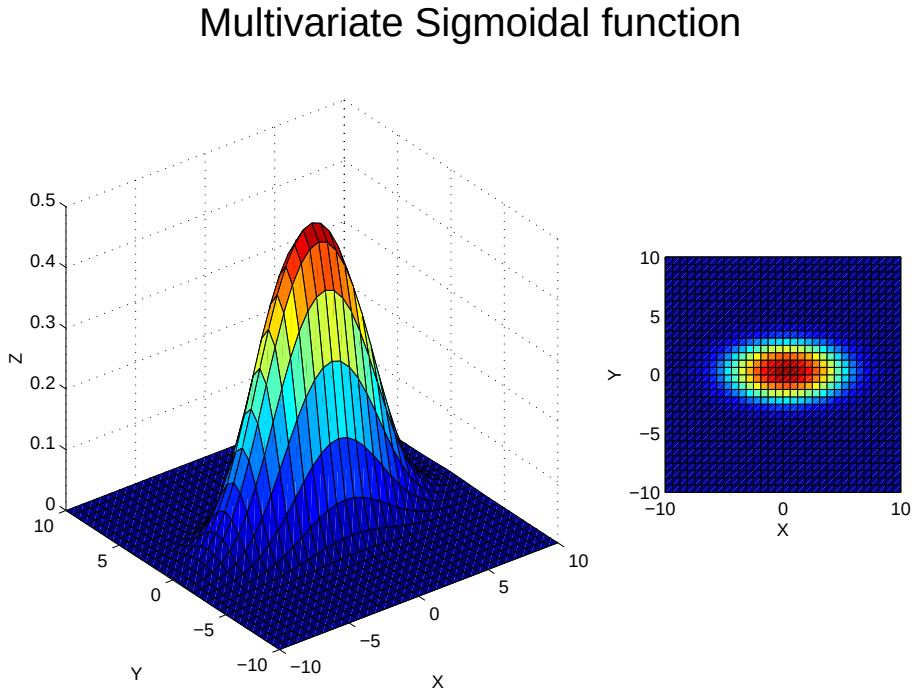

Figure 20: Multivariate Sigmoidal function (Eq. 79).

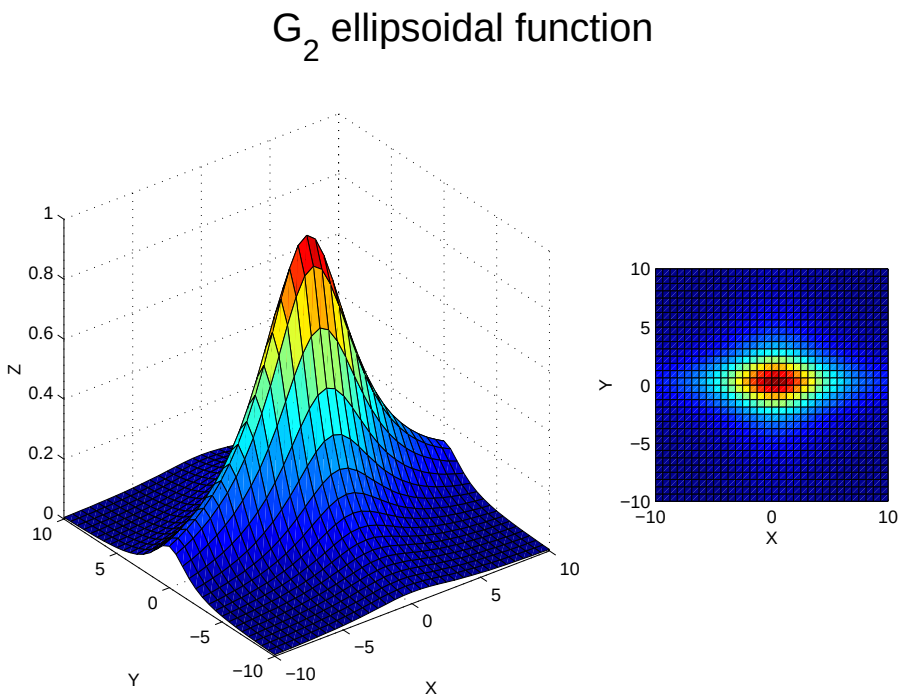

Figure 21: $\bar{G_2}$ function (Eq. 82).

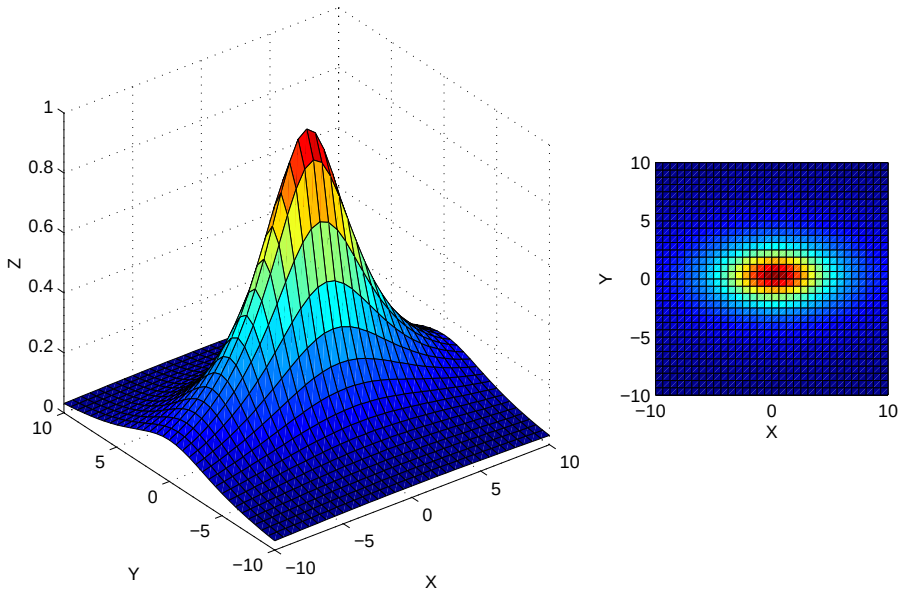

Figure 22: $\bar{G_3}$ function (Eq. 83).

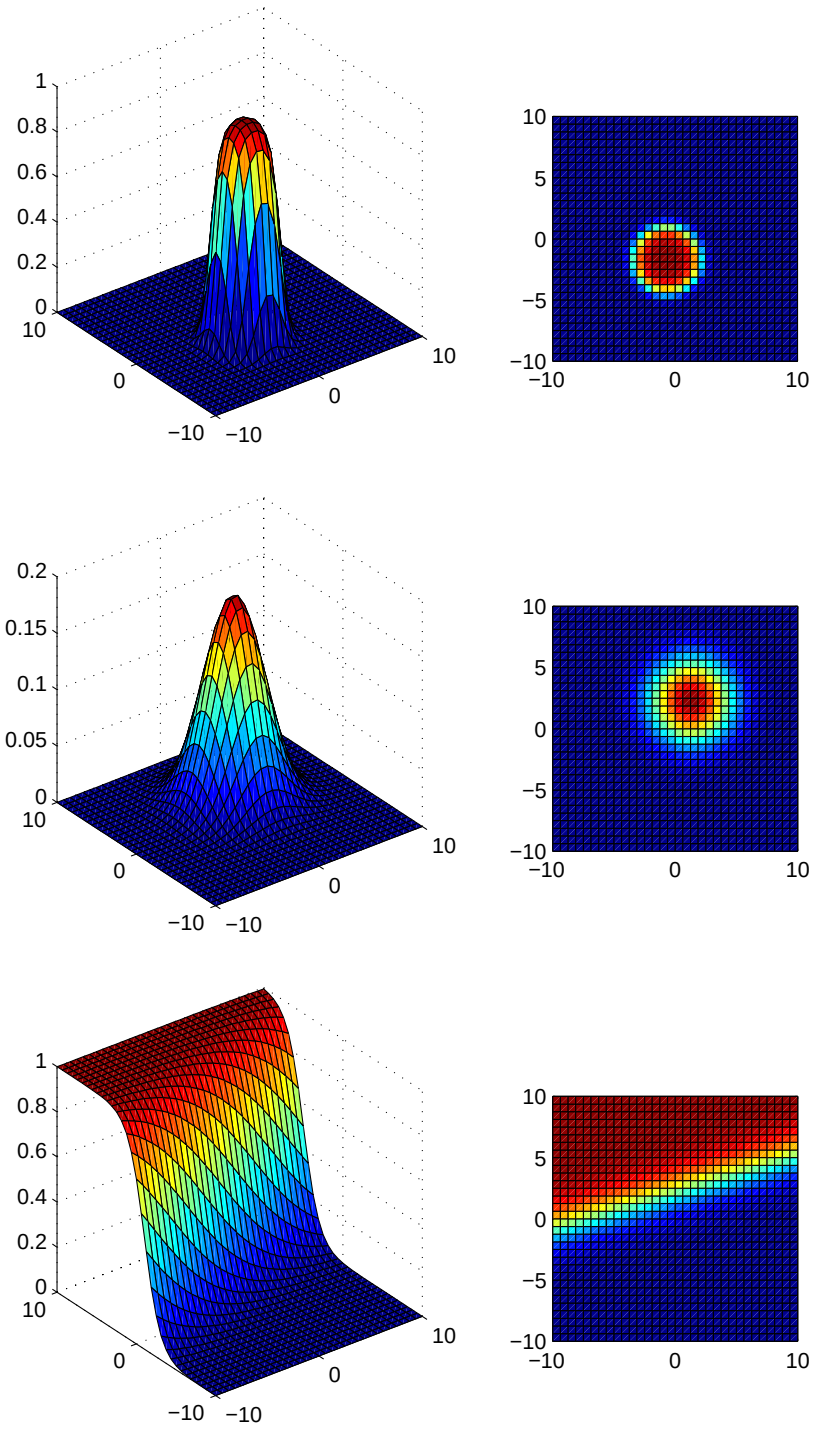

Figure 23: Riddella function (Eq. 85).

$$A_R(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{N} w_i x_i + w_{N+1} \sum_{i=1}^{N} x_i^2 \tag{85}$$

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

$$\begin{aligned} A_R(\mathbf{x};\mathbf{w}) &= d_c(\mathbf{x};\mathbf{c}) = (||\mathbf{x}-\mathbf{c}||^2 - \theta) w_{N+1}; \\ c_i &= -w_i/2w_{N+1}; \quad \theta = \frac{1}{w_{N+1}} \left(\sum_{i=1}^{N} \frac{w_i^2}{4w_{N+1}^2} - w_0 \right) \end{aligned} \tag{86}$$

Ridella *et al.* [47] 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 types of circular units have been used by Kirby and Miranda [77]. In their implementation two sigmoidal units are coupled together and their output is restricted to lie on a unit circle.

Dorffner [13] proposed *conic section* transfer functions as a unified framework for the 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 (Fig. 24):

$$\begin{aligned} A_C(\mathbf{x};\mathbf{w},\mathbf{t},\omega) &= I(\mathbf{x}-\mathbf{t};\mathbf{w}) + \omega D(\mathbf{x}-\mathbf{t}) \\ &= \sum_{i=1}^{N+1} w_i (x_i - t_i) + \omega \sqrt{\sum_{i=1}^{N+1} (x_i - t_i)^2} \end{aligned} \tag{87}$$

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 (see Fig. 25):

$$C_{GL1}(\mathbf{x};\mathbf{w},\mathbf{t},\alpha,\theta) = \frac{1}{1+A_{GL1}} = \frac{1}{1+(I(\mathbf{x};\mathbf{w}) + \alpha D(\mathbf{x};\mathbf{t}))^2} \tag{88}$$

or

$$C_{GL2}(\mathbf{x};\mathbf{w},\mathbf{t},\alpha,\theta) = \frac{1}{1+A_{GL2}} = \frac{1}{1+\alpha I^2(\mathbf{x};\mathbf{w}) + \beta D^2(\mathbf{x};\mathbf{t})} \tag{89}$$

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 these functions are nonseparable .

5.6 Bicentral functions

Sigmoidal functions may be combined into a *window* type localized functions in several ways. Two simple window-type functions are constructed as the difference of two sigmoids, $\sigma(x) - \sigma(x-\theta)$ or the product of pairs of sigmoidal functions $\sigma(x)(1-\sigma(x))$ for each dimension. After normalization the two forms are identical:

$$\frac{\sigma(x+b)(1-\sigma(x-b))}{\sigma(b)(1-\sigma(-b))} = \frac{\sigma(x+b)-\sigma(x-b)}{\sigma(b)-\sigma(-b)} \tag{90}$$

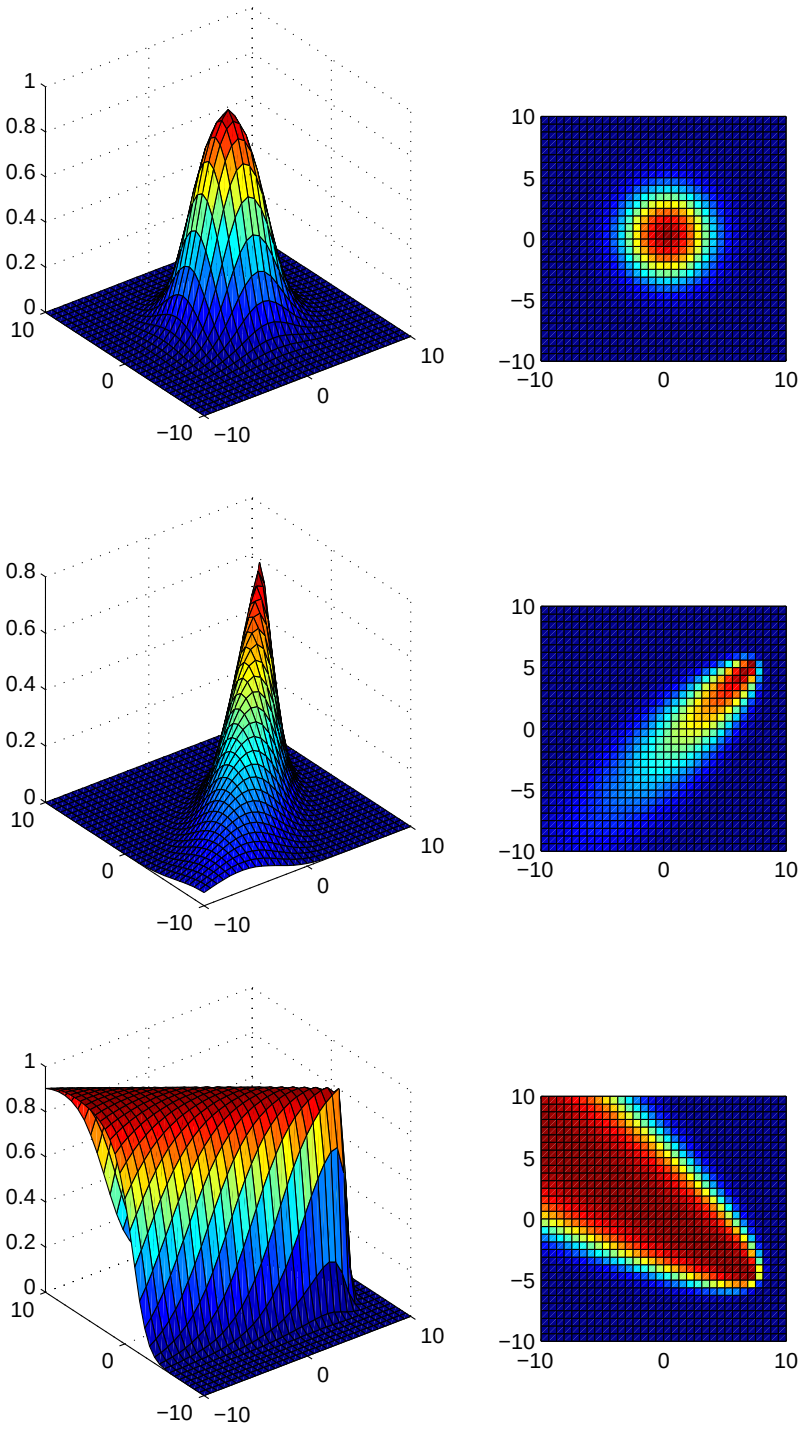

Figure 24: Conical function (Eq. 87).

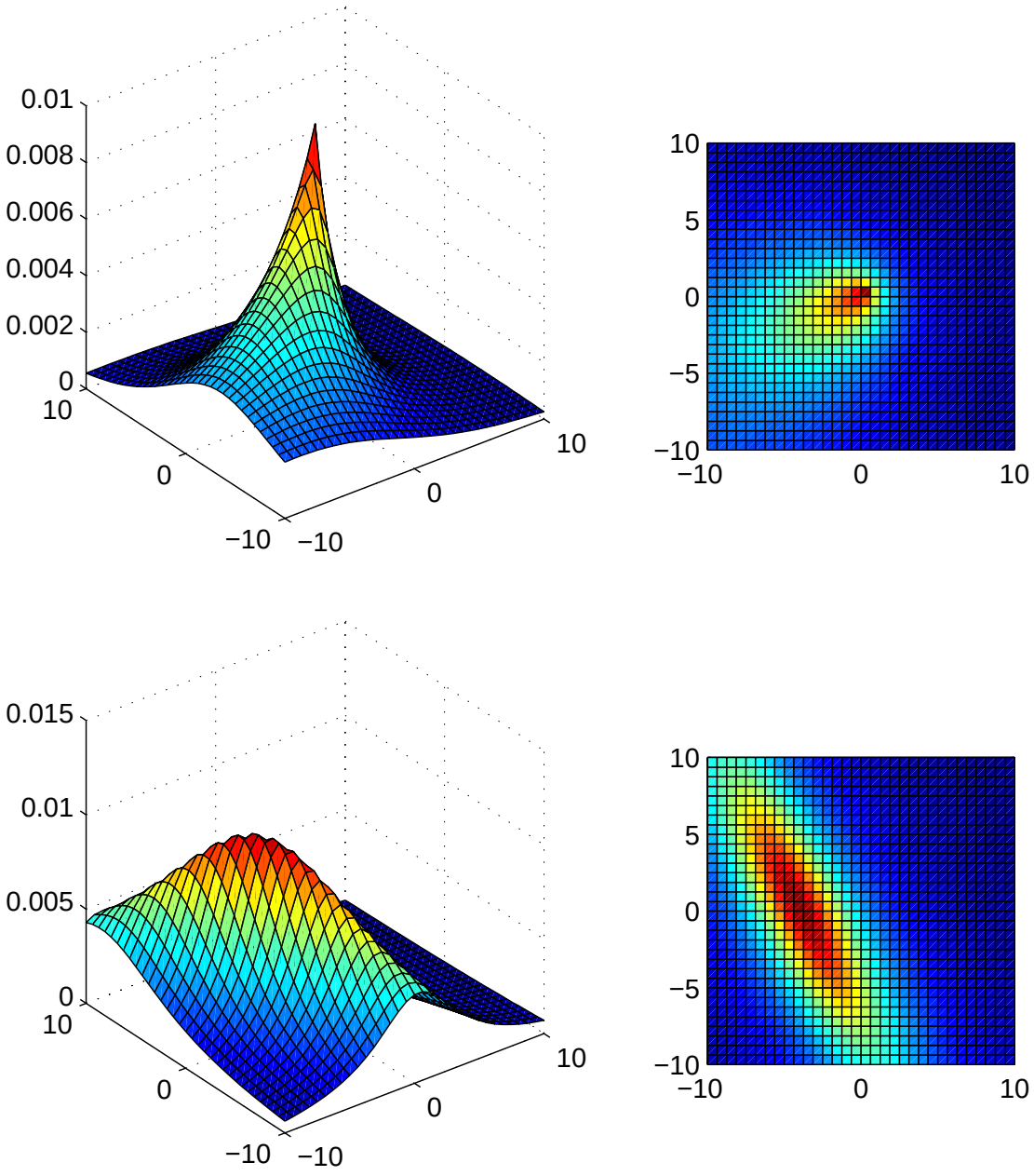

Figure 25: Approximated Gaussian combined with the Lorentzian function (Eq. 88 and 89).

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 (Fig. 26):

$$\begin{aligned} Bi(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) &= \prod_{i=1}^{N} \sigma(A1_i^+)\ (1-\sigma(A1_i^-)) \\ &= \prod_{i=1}^{N} \sigma(e^{s_i}\cdot(x_i-t_i+e^{b_i}))\ (1-\sigma(e^{s_i}\cdot(x_i-t_i-e^{b_i}))) \end{aligned} \tag{91}$$

where $\sigma(x)$ is a logistic function (Eq. 5). 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 components of two centers are used, $t_i+e^{b_i}$ and $t_i-e^{b_i}$, therefore we have called these functions previously *biradial functions* [78], but perhaps the name *bicentral* is more appropriate. Product form leads to well-localized convex contours of bicentral functions. 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).

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 contours are obtained, thus reducing the number of adaptive units in the network.

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

$$\begin{aligned} SBi(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) &= \prod_{i=1}^{N} (\alpha+\sigma(A1_i^+))\ (1-\beta\sigma(A1_i^-)) \\ &= \prod_{i=1}^{N} (\alpha+\sigma(e^{s_i}\cdot(x_i-t_i+e^{b_i})))\ (1-\beta\sigma(e^{s_i}\cdot(x_i-t_i-e^{b_i}))) \end{aligned} \tag{92}$$

This function does not vanish for large $|x|$, for $\alpha=0$, $\beta=1$ it is identical to the bicentral 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).

Bicentral functions with independent slopes. Another possibility to control contours of constant value of bicentral functions is to use independent slopes (see Fig. 27):

$$\begin{aligned} Bi2s(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) &= \prod_{i=1}^{N} \sigma(A2_i^+)\ (1-\sigma(A2_i^-)) \\ &= \prod_{i=1}^{N} \sigma(e^{s_i}\cdot(x_i-t_i+e^{b_i}))\ (1-\sigma(e^{s_i'}\cdot(x_i-t_i-e^{b_i}))) \end{aligned} \tag{93}$$

Using small slope s_i and/or s_i' the bicentral function may delocalize or stretch to *left* and/or *right* in any dimension. This allows creation of such contours of transfer functions as half-infinite chanel, half-hyper ellipsoidal, soft triangular, etc. Although the costs of using this function is a bit higher than of the bicentral function (each function requires $4N$ parameters) more flexible decision borders are produced.

Bicentral functions with rotation. The bicentral functions proposed above contain $3N$ parameters per unit and may represent quite complex decision borders. Semi-bicentral functions and bicentral functions with independent slopes provide local and non-local units in one network. The next step towards even greater flexibility requires individual rotation of contours provided by each unit [78, 79]. Of course one could introduce a rotation matrix operating on the

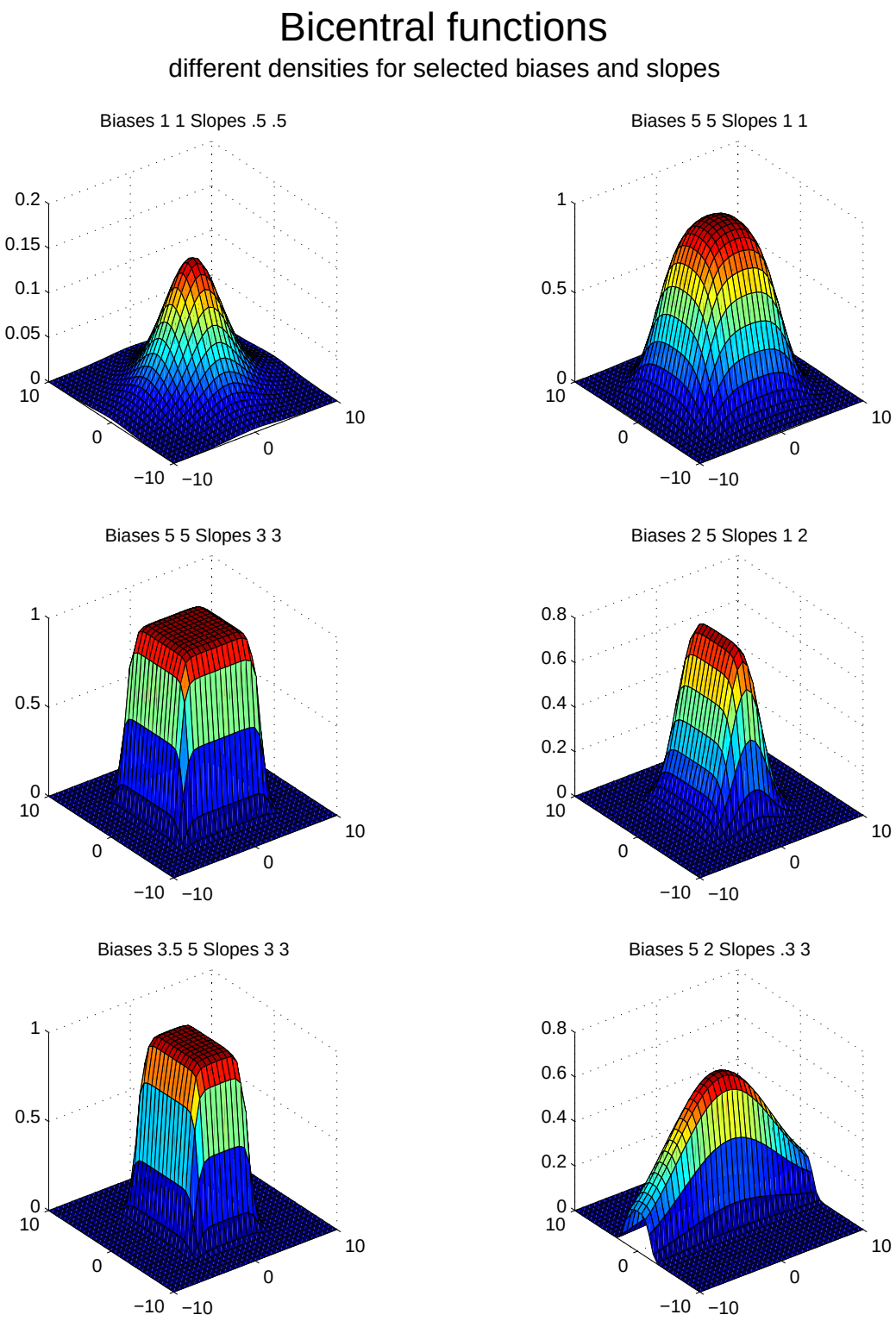

Figure 26: A few shapes of the bicentral functions (Eq. 91).

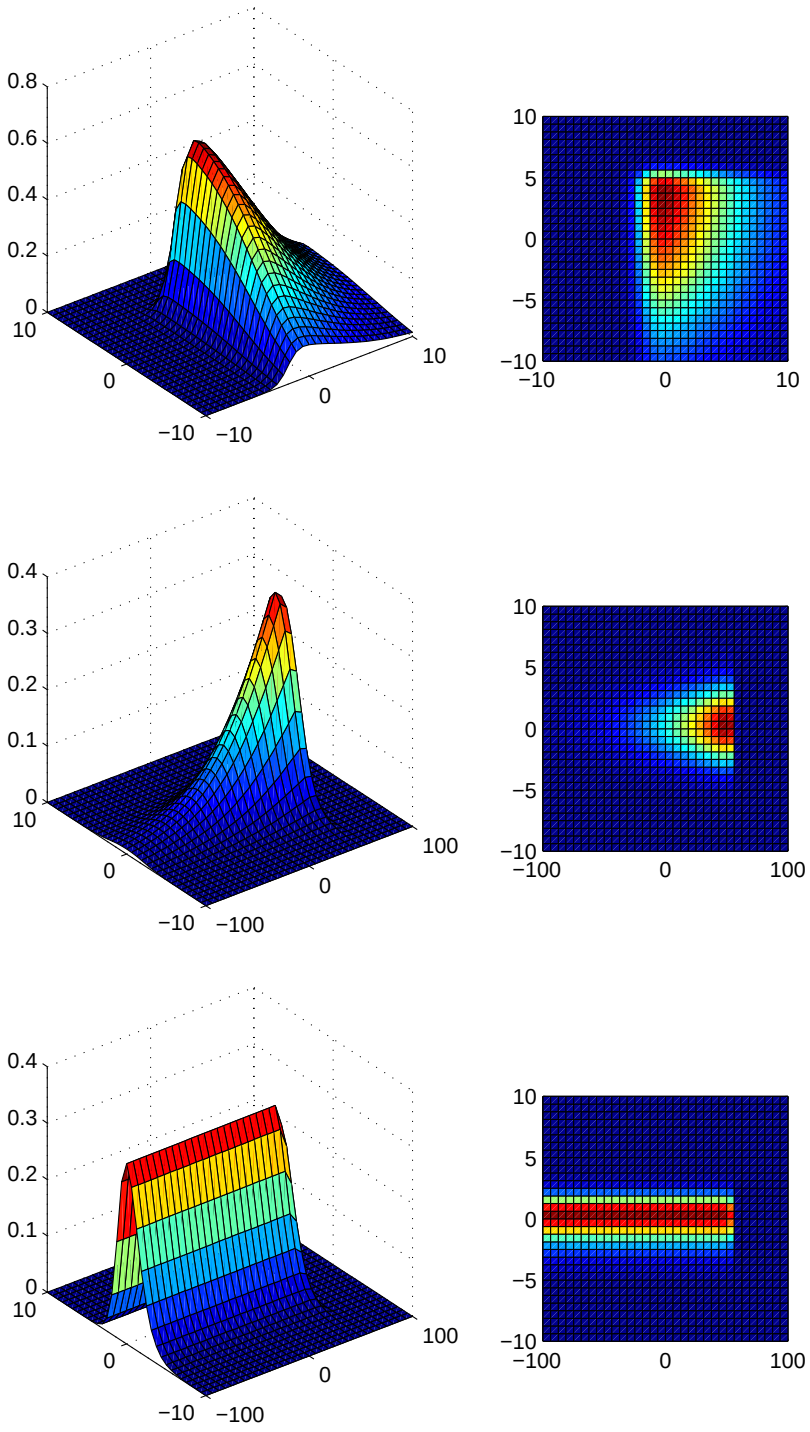

Figure 27: Bicentral functions with two slopes (Eq. 93).

inputs **Rx**, but in practice it is very hard to parametrize this $N \times N$ matrix with $N-1$ independent angles (for example, with Euler's angles) and to calculate all derivatives necessary for backpropagation training procedure. We have found two ways to obtain rotated contours in all dimensions using transfer functions with just N additional parameters per neuron. In the first approach, a product form of the combination of sigmoids is used (see Fig. 28)

$$\begin{aligned} C_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{R}) &= \prod_i^N \Big(\sigma(A3_i^+) - \sigma(A3_i^-)\Big) \qquad (94)\\ &= \prod_i^N \Big(\sigma(\mathbf{R}_i\mathbf{x}+t_i) - \sigma(\mathbf{R}_i\mathbf{x}+t_i')\Big) \\ SC_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{p},\mathbf{r},\mathbf{R}) &= \prod_i^N \Big(p_i\cdot\sigma(A3_i^+) + r_i\cdot\sigma(A3_i^-)\Big) \qquad (95)\\ &= \prod_i^N \Big(p_i\cdot\sigma(\mathbf{R}_i\mathbf{x}+t_i) + r_i\cdot\sigma(\mathbf{R}_i\mathbf{x}+t_i')\Big) \end{aligned}$$

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

$$\mathbf{R} = \begin{bmatrix} s_1 & \alpha_1 & 0 & & \cdots & 0 \\ 0 & s_2 & \alpha_2 & 0 & & \\ & & & & & \\ \vdots & & & \ddots & & \vdots \\ & & & & s_{N-1} & \alpha_{N-1} \\ 0 & \cdots & & & 0 & s_N \end{bmatrix} \qquad (96)$$

If $p_i = 1$ and $r_i = -1$ then SC_P function is localized and has similar contours as the bicentral 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 sum of a *window-type* combinations of sigmoids $L(x;t,t') = \sigma(x+t) - \sigma(x+t')$ in $N-1$ dimensions is used and the last combination is rotated by a vector **k**:

$$C_K(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{w},\mathbf{k}) = \sum_{i=1}^{N-1} w_i L(x_i,t_i,t_i') + w_N L(\mathbf{kx},t,t') \qquad (97)$$

The last term $L(\mathbf{kx},t,t')$ has contours of constant value extending in perpendicular direction to the **k** vector. Treating $C_K(\cdot)$ as the activation function and using a sigmoidal output function with a proper threshold leaves non-zero values only in the direction perpendicular to **k**. An alternative is to use the product form:

$$C_{PK}(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{k}) = L(\mathbf{kx},t,t') \prod_{i=1}^{N-1} L(x_i,t_i,t_i') \qquad (98)$$

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.

Bicentral functions with rotations (as well as multivariate Gaussian functions with rotation) have been implemented so far only in two neural network models, the Feature Space Mapping [65, 79] and the IncNet [80, 81, 82].

Bicentral functions with rotation and two slopes. The most complex bicentral function is obtained by combining rotations with two independent slopes (see Fig. 29):

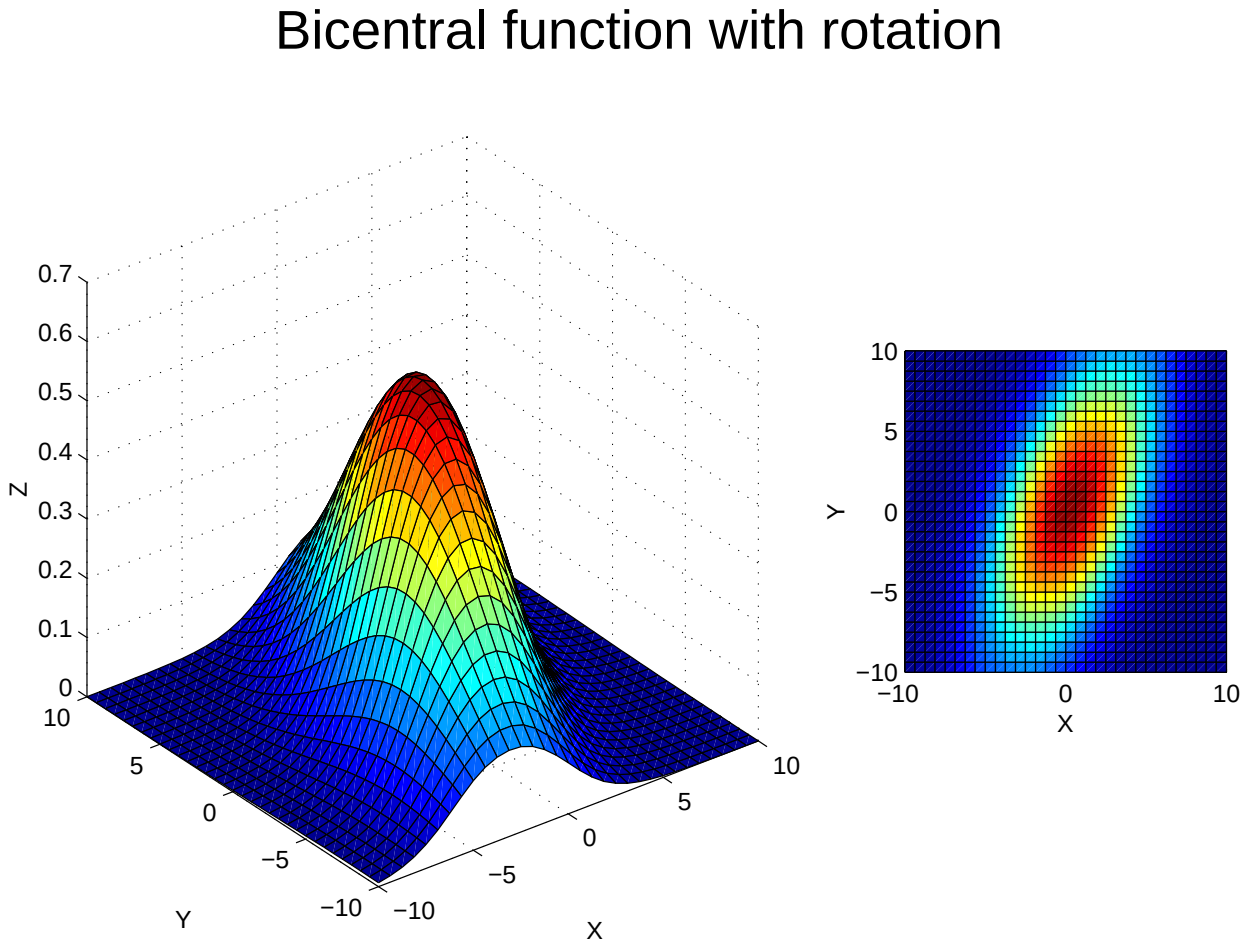

Figure 28: Bicentral functions with rotation (Eq. 94).

$$
\begin{aligned}
BiR2s(\mathbf{x};\mathbf{t},\mathbf{t}',\alpha) &= \prod_i^N \sigma(A4_i^+)(1-\sigma(A4_i^-)) \\
&= \prod_i^N \sigma(s_i(x_i+\alpha_i x_{i+1}-t_i+b_i))(1-\sigma(s_i'(x_i+\alpha_i x_{i+1}-t_i-b_i)))
\end{aligned}
\tag{99}
$$

where $\alpha_1,\ldots,\alpha_{N-1}$ define the rotation and $x_{N+1}=0$ and $\alpha_N=0$ is assumed. This transfer function can be local or semi-local and may rotate in any direction, therefore it is computationally more expensive, using $5N$ adaptive parameters per function.

An important advantage of the bicentral functions comes from their separability. Sigmoidal functions are not separable and among radial basis functions only Gaussians are separable. Although bicentral functions are made from sigmoids they are separable because products of pairs of sigmoids for each dimension are used. Separability enables analysis of each dimension or a subspace of the input data independently: 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 and allows to implement associative memories using feedforward networks [65, 79].

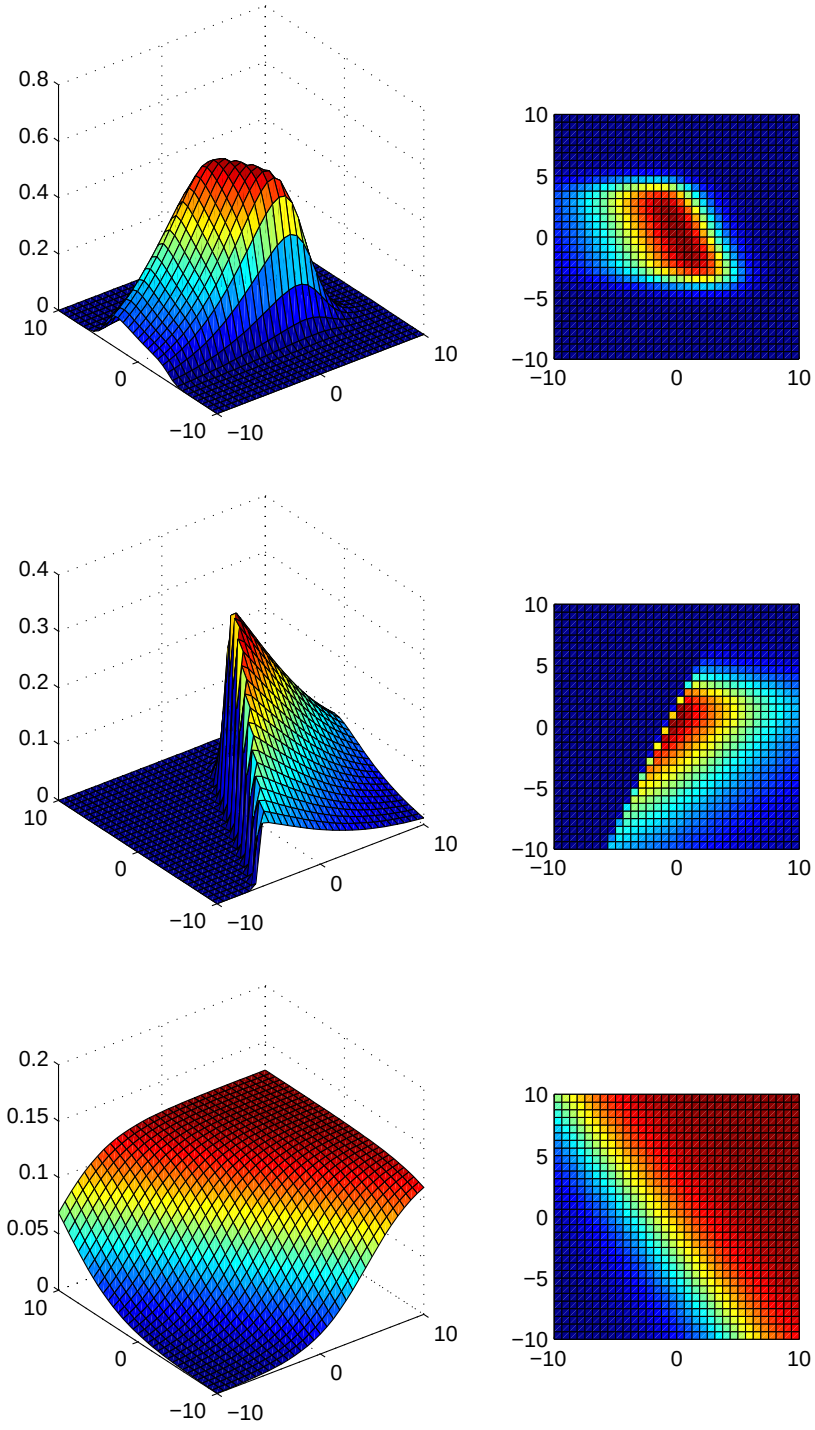

Figure 29: Bicentral functions with rotation and two slopes (Eq. 99).

Bicentral transfer functions may also be used for *logical rule extraction* using the Feature Space Mapping network (FSM network, a model similar to RBF but based on separable functions) [83, 84]. Logical interpretation of the function realized by this neural network is possible if instead of hyperellipsoidal densities cuboidal densities are used. In case of sigmoidal and bicentral transfer functions sufficiently large values of the slopes are needed, changing graded sigmoidal functions into step functions and bicentral functions into cuboidal (rectangular) functions. There are several ways to enforce large slopes of the transfer functions, for example by adding penalty terms to the error function. 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 relevant inputs are removed. Using these ideas we have obtained very good results in applications to rule extraction from data [84].

A comparison of different transfer functions is presented in Table 1. in the first column the name of the transfer function is given; the second column shows the equation number defining the function; the third column shows the type of activation function. In the next column the number of adaptive parameters for d-dimensional inputs are given. For the multistep function k is the number of different steps (usually not treated as an adaptive parameter). Local or non-local character of functions is noted in the fifth column – some functions may be either local or non-local, depending on their parameters. Next column shows the type of adaptive parameters: **w** are weight-type linear parameters, θ are thresholds, **t** are centers, **b** and b play the role of dispersions or feature scaling factors, **s** determine slopes, **R** are parameters determining rotation, and o, O are additional parameters. The last two columns note whether the function is separable (Y), symmetric (S), antisymmetric (A) or has no symmetry (N).

6 NONLINEAR TRANSFORMATION OF INPUTS.

So far various transfer functions were discussed providing flexible decision borders for complex problems. Another approach has been taken in the functional link networks of Pao [11]. New input features, obtained by transformation of original inputs, are added as new inputs to a standard MLP network. Such transformations are performed also in the pattern recognition methods [85] and newer work on Support Vector Machines [86]. Transfer functions are not changed directly but adding additional inputs has a strong influence on decision borders. The circular units used by Ridella (Eq. 85) may either be presented as a new type of an activation function or as an additional input receiving the sum of squared values of original inputs.

The distance-based activation functions have been combined with the sigmoidal output functions in the multilayer perceptron models only very recently [75, 76]. Neural realization of minimal distance methods, including some novel transfer functions, is discussed in [87, 88]. For inputs normalized to $||\mathbf{x}|| = ||\mathbf{w}|| = 1$ the fan-in activation of neurons is strongest for input vectors **x** that are close to **w** on a unit sphere. In general the activation of a neuron may be written as:

$$\mathbf{w}\cdot\mathbf{x} = \frac{1}{2}\left(||\mathbf{w}||^2 + ||\mathbf{x}||^2 - ||\mathbf{w}-\mathbf{x}||^2\right) \tag{100}$$

For normalized input vectors sigmoidal functions (or any other monotonically growing transfer functions) may therefore be written in the form:

$$\sigma(\mathbf{w}\cdot\mathbf{x}+\theta) = \sigma(d_0 - D^2(\mathbf{w},\mathbf{x})) \tag{101}$$

where $D^2(\mathbf{w},\mathbf{x}) = \frac{1}{2}||\mathbf{w}-\mathbf{x}||^2$. This distance function evaluates the influence of the reference vectors **w** on the classification probability $p(C_i|\mathbf{x};\mathbf{w})$. Transfer function $f(D) = \sigma(d_0 - D^2(\mathbf{w},\mathbf{x}))$ monotonically decreases as a function of distance, with flat plateau for small distances D, reaching the value of 0.5 for $D^2(\mathbf{w},\mathbf{x}) = d_0$ and approaching zero for larger distances. For normalized **x** but arbitrary **w** the sigmoid arguments belong to the $[\theta - |\mathbf{w}|, \theta + |\mathbf{w}|]$ interval. A unipolar sigmoid has its maximum curvature around ± 2.4, therefore smaller thresholds θ and absolute weight values

Function	Eq.	Activation	Number of parameters	Local or nonlocal	Adaptation of	Separability	Symetric
Heaviside	(2)	I	$d+1$	NL	$\mathbf{w},\theta$		A
multistep	(3)	I	$d+k$	NL	$\mathbf{w},\Theta$		A
Semi-linear	(4)	I	$d+2$	NL	$\mathbf{w},\theta$		A
Logistic	(5)	I	$d+1$	NL	$\mathbf{w},\theta$		A
tanh, arctan	(46)	I	$d+1$	NL	$\mathbf{w},\theta$		A
s_1	(48)	I	$d+1$	NL	$\mathbf{w},\theta$		A
s_2	(49)	I	$d+1$	NL	$\mathbf{w},\theta$		A
s_3	(50)	I	$d+1$	NL	$\mathbf{w},\theta$		A
s_4	(51)	I	$d+1$	NL	$\mathbf{w},\theta$		A
Radial coordinate	(56)	D	d	NL	$\mathbf{t},b$		S
Multiquadratics	(58)	D	$d+2$	L+NL	$\mathbf{t},b,o$		S
Thin-plate spline	(60)	D	$d+1$	NL	$\mathbf{t},b$		S
Gaussian	(61)	D	$d+1$	L	$\mathbf{t},b$	Y	S
$G_1 = 2-2\sigma(r^2)$	(62)	D	$d+1$	L	$\mathbf{t},b$		S
$G_2 = \tanh(r^2)$	(63)	D	$d+1$	L	$\mathbf{t},b$		S
G_3	(64)	D	$d+1$	L	$\mathbf{t},b$		S
G_4	(65)	D	$d+1$	L	$\mathbf{t},b$		S
RCBSpline	(66)	D	$d+1$	L	$\mathbf{t},b$		S
RQBSpline	(67)	D	$d+1$	L	$\mathbf{t},b$		S
Gaussian bar	(76)	D_i	$3d$	L	$\mathbf{t},\mathbf{b},O$	Y	S
Sigmoidal bar	(77)	D_i	$3d$	L	$\mathbf{t},\mathbf{b},O$	Y	S
Multivariate gaussian	(78)	D	$2d$	L	$\mathbf{t},\mathbf{b}$	Y	S
Multivariate sigmoid	(79)	D	$2d$	L	$\mathbf{t},\mathbf{b}$		S
$\bar{G}_2$	(82)	D_i	$2d$	L	$\mathbf{t},\mathbf{b}$	Y	S
$\bar{G}_3$	(83)	D	$2d$	L	$\mathbf{t},\mathbf{b}$		S
Lorentzian	(71)	I	$d+1$	NL	$\mathbf{t},\mathbf{s}$		N
Tensor-prod.	(72)	D_i	$2d+1$	NL	$\mathbf{w},\theta$		N
G_R	(84)	D	$2d$	NL	$\mathbf{t},\mathbf{b}$		N
Ridella	(85)	A_R	$d+3$	L+NL	$\mathbf{w},\theta$		S+N
Conical	(87)	A_C	$2d+2$	L+NL	$\mathbf{t},\mathbf{w},\theta,o$		S+N
C_{GL1}	(88)	A_{GL1}	$2d+3$	L+NL	$\mathbf{t},\mathbf{w},\theta,o$		S+N
C_{GL2}	(89)	A_{GL2}	$2d+4$	L+NL	$\mathbf{t},\mathbf{w},\theta,o$		S+N
Bicentral	(91)	$A1$	$3d$	L	$\mathbf{t},\mathbf{b},\mathbf{s}$	Y	S
Semi-bicentral	(92)	$A2$	$5d$	L+NL	$\mathbf{t},\mathbf{b},\mathbf{s},O$	Y	S+N
Bicentral 2 slopes	(93)	$A2$	$4d$	L+NL	$\mathbf{t},\mathbf{b},\mathbf{s}$	Y	S+N
Bicentral rotation	(94)	$A3$	$4d-1$	L	$\mathbf{t},\mathbf{b},\mathbf{s},R$	Y/N	S
Semi-bicentral rotation	(95)	$A3$	$6d-1$	L+NL	$\mathbf{t},\mathbf{b},\mathbf{s},R,O$	Y/N	S+N
C_K rotation	(97)	$A1,A1(\mathbf{kx})$	$4d$	L	$\mathbf{t},\mathbf{b},\mathbf{s},R$	Y/N	S
C_{PK} rotation	(98)	$A1,A1(\mathbf{kx})$	$4d$	L	$\mathbf{t},\mathbf{b},\mathbf{s},R$	Y/N	S
Bicentral rotation 2 slopes	(99)	$A4$	$5d-1$	L+NL	$\mathbf{t},\mathbf{b},\mathbf{s},R$	Y/N	S+N

Table 1: Comparison of different transfer functions. See text for explanation of symbols used.

$|\mathbf{w}|$ mean that the network operates in an almost linear regime and thus smoothing the network approximation to the training data. Small weights are enforced in regularization methods by adding penalty terms to the error function.

The interpretation here is that MLP networks use sigmoidal functions to estimate the influence of weight vectors according to distance between the weight and the training vectors, combining many such estimations to compute the final output. Changing the distance function in Eq. (101) from the square of the Euclidean distance to some other distance measures distance-based neural networks (D-MLP networks, [75]), are defined.

Another possibility to replace the fan-in activation by distance functions is to write the weighted product in the form:

$$\sigma(\mathbf{w}\cdot\mathbf{x}) \quad = \quad \sigma\left(\frac{1}{4}(||\mathbf{w}+\mathbf{x}||^2 - ||\mathbf{w}-\mathbf{x}||^2)\right) \tag{102}$$

and replace the Euclidean norm by Minkovsky's or other type of norms.

Instead of deriving the backpropagation equations for the transfer functions with non-Euclidean distances providing non-planar decision borders one may achieve similar result using a standard MLP network in the extended feature space, where the new x_r component is determined by the normalization condition using the desired metric [76]. Extended vectors $||(\mathbf{x},x_r)||_D$ are renormalized using the metric function $D(\cdot)$, placing them on a unit sphere defined by this metric. The influence of input renormalization (using Minkovsky distance functions) on the shapes of decision borders is illustrated in Fig. 30 for the classical Iris flowers dataset (only the last two input features, x_3 and x_4 are shown, for description of the data cf. [89]). Dramatic changes in the shapes of decision borders for different Minkovsky metrices are observed. Using squared Euclidean metric with $\sigma(d_0 - D^2(\mathbf{x},\mathbf{t}))$ transfer functions and $w_3 = 0$ the standard MLP solution is obtained. Euclidean case corresponds to circular decision borders, the city block metric $\alpha = 1$ gives sharp, romboidal shapes, for large α almost rectangular decision borders are obtained (an approximation using logical rules is in this case straightforward) while for small α hypocycloidal shapes are created.

Thus adding new input features (more than one may be added) obtained from nonlinear transformation is an alternative method to increase flexibility of decision borders. If the neural model used allows for estimation of its reliability (for example by inspecting the outputs of localized nodes) one may use several networks with different preprocessing of inputs and either treat them as an ensemble or select the answer from the network which is estimated to be most reliable.

7 DISCUSSION AND POSSIBLE EXTENSIONS

We have presented an overview of different transfer functions used in neural network models and proposed several new combinations of activation and output functions suitable for this purpose. From the geometrical point of view learning requires approximation of complicated posterior probability decision borders or probability density contours. In the process of neural networks training 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, bicentral or simplified Lorentzian/Gaussian functions lead to more compact networks that may learn faster and generalize better. These functions use mixed activations to unify the distance-based, localized paradigm with activation functions that are quadratic in inputs, and the non-local approximations based on discriminant functions that use only linear activations.

Unfortunately little is known about advantages and disadvantages of different transfer functions presented in this paper. Many of them have never been used in neural networks so far and almost none are available in public domain neural software. Bicentral and rotated transfer functions ($C_P(\cdot)$, $C_S(\cdot)$) are flexible but still rather simple, therefore we have used them in the FSM [65, 90, 79], IncNet

[80, 81, 82, 91] and other neural models, obtaining very good results. Fair comparison of transfer functions would be very difficult, if not impossible, because results depend on the methods, learning algorithms, initialization, architectures and datasets used. Empirical large scale comparisons of classification methods were attempted in the Statlog project [17], but we still do not know how to characterize datasets to determine which method to use for a given dataset. Similarly we do not know which transfer functions should be used for different datasets. This does not

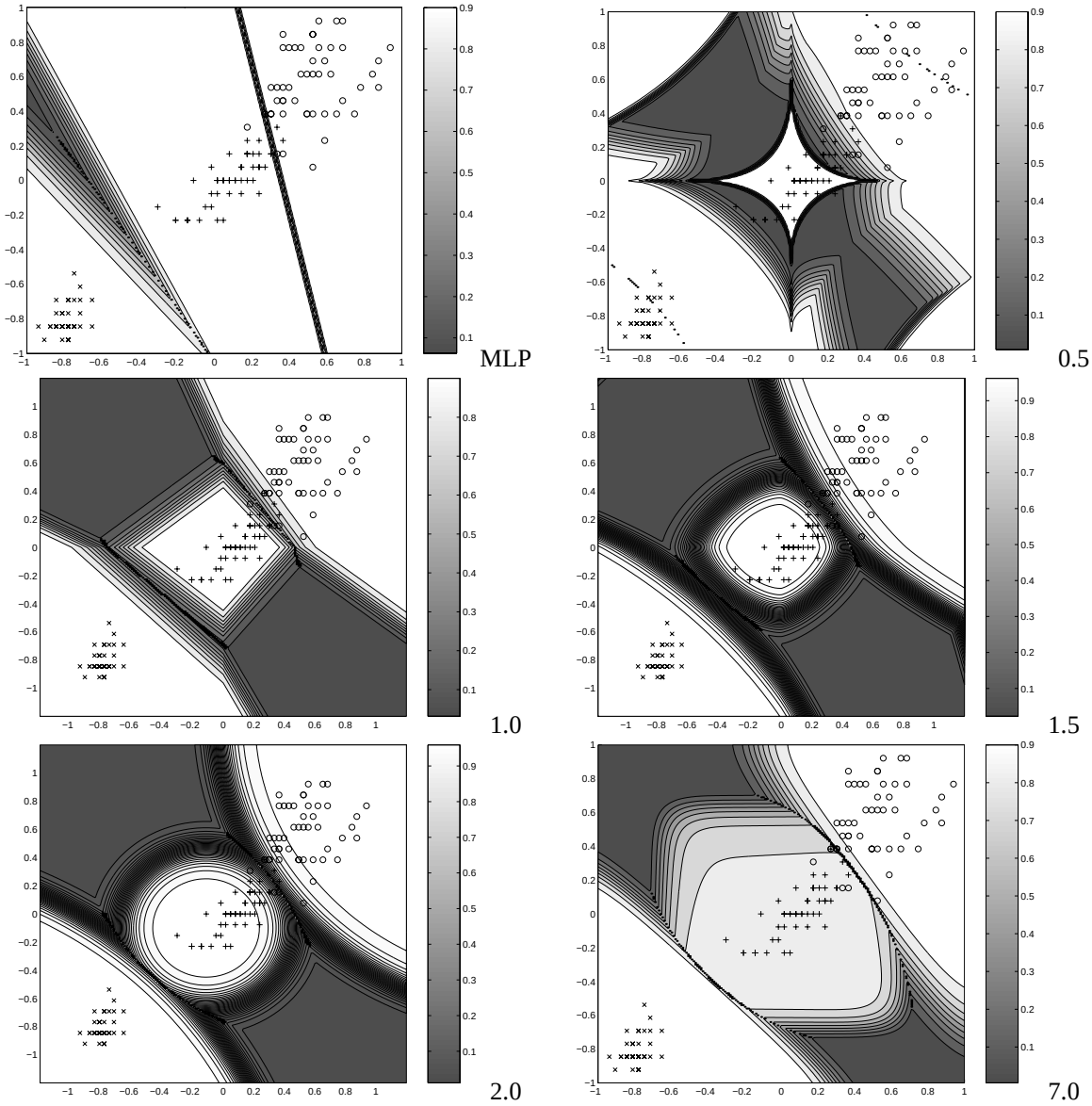

Figure 30: Shapes of decision borders in the Iris case for standard MLP (4 neurons, 2 inputs) solution and for MLP (3 neurons, 3 inputs) using the data vectors renormalized with Minkovsky metric, $\alpha = 0.5, 1.0, 1.5, 2.0$ and 7.0.

mean that we should always be satisfied with sigmoidal functions any more than that we should be satisfied with a single classification model. Many transfer functions may be tried with a single neural model, keeping all other factors constant, therefore at least partial empirical comparison of the effectiveness of transfer functions should be possible.

Although the importance of flexible description of decision borders seems to be rather obvious, the important role of the transfer functions is frequently overlooked. There is a tradeoff between flexibility of a single processing unit, increasing with the number of adjustable parameters, and the number of units needed to solve the problem, decreasing with flexibility of individual units. The complexity of the training process of the whole network should be minimized, but what is the optimal balance between the number and the complexity of units is not known. Papers describing novel transfer functions quoted in this survey contain many interesting results, although a systematic comparison of transfer functions has not been done. Very few neural network simulators use transfer functions that are not sigmoidal or Gaussian, therefore little experience has been gathered. We believe that neural transfer functions deserve more attention.

REFERENCES

[1] S. Haykin, *Neural Networks - A Comprehensive Foundation*, Maxwell MacMillian Int., New York, 1994.

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

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

[4] 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, San Diego, 1989.

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

[6] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[7] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[8] E. J. Hartman, J. D. Keeler, and J. M. Kowalski, "Layered neural networks with gaussian hidden units as universal approximations", *Neural Computation*, vol. 2, no. 2, pp. 210–215, 1990.

[9] J. Park and I. W. Sandberg, "Universal approximation using radial basis function networks", *Neural Computation*, vol. 3, no. 2, pp. 246–257, 1991.

[10] E. Hartman and J. D. Keeler, "Predicting the future: Advantages of semilocal units", *Neural Computation*, vol. 3, no. 4, pp. 566–578, 1991.

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

[12] H. Leung and S. Haykin, "Rational neural networks", *Neural Computation*, vol. 5, no. 6, pp. 928–938, 1993.

[13] G. Dorffner, "A unified framework for MLPs and RBFNs: Introducing conic section function networks", *Cybernetics and Systems*, vol. 25, no. 4, pp. 511–554, 1994.

[14] B. G. Giraud, A. Lapedes, L. C. Liu, and J. C. Lemm, "Lorentzian neural nets", *Neural Networks*, vol. 8, no. 5, pp. 757–767, 1995.

[15] T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification", in *IEEE PAMI 18*, 1996, pp. 607–616.

[16] J. H. Friedman, "Flexible metric nearest neighbor classification", Technical report, Department of Statistics, Stanford University, 1994.

[17] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine learning, neural and statistical classification*, Elis Horwood, London, 1994.

[18] J. A. Anderson, *An Introduction to Neural Networks*, Bradford Book, 1995.

[19] W. Maas, "Lower bounds for the computational power of networks of spiking neurons", *Neural Computations*, vol. 8, pp. 1–40, 1996.

[20] T. Miki, M. Shimono, and T. Yamakawa, "A chaos hardware unit employing the peak point modulation", in *Proceedings of the International Symposium on Nonlinear Theory and its Applications*, Las Vegas, Nevada, 1995. pp. 25–30.

[21] T. Yamakawa, M. Shimono, and T. Miki, "Design criteria for robust associative memory employing non-equilibrium network.", in *Proceedings of the 4th International Conference on Soft Computing (IIZUKA'96)*, Iizuka, 1996. pp. 688–691.

[22] C. C. Hsu, D. Gubovic, M. E. Zaghloul, and H. H. Szu, "Chaotic neuron models and their VLSI implementations", *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1339–1350, Nov. 1996.

[23] K. Kobayasji, "On the capacity of neuron with a non-monotone output function", *Network*, vol. 2, pp. 237–243, 1991.

[24] M. Morita, "Associative memory with nonmonotone dynamics", *Neural Networks*, vol. 6, pp. 115–126, 1993.

[25] S. Yoshizawa, M. Morita, and S. Amari, "Capacity of associative memory using a nonmonotonic neuron model", *Neural Networks*, vol. 6, pp. 167–176, 1993.

[26] H. F. Yanai and S. Amari, "A theory of neural net with non-monotone neurons", in *Proceedings of IEEE International Conference on Neural Networks*, 1993, pp. 1385–1390.

[27] H. F. Yanai and S. Amari, "Auto-associative memory with two-stage dynamics of non-monotonic neurons", *IEEE Transactions on Neural Networks*, 1998, (submitted).

[28] H. F. Yanai and Y. Sawada, "Associative memory network composed of neurons with histeretic property", *Neural Networks*, vol. 3, pp. 223–228, 1990.

[29] T. Kasahara and M. Nakagawa, "A study of association model with periodic chaos neurons", *Journal of Physical Society*, vol. 64, pp. 4964–4977, 1995.

[30] M. Nakagawa, "An artificial neuron model with a periodic activation function", *Journal of Physical Society*, vol. 64, pp. 1023–1031, 1995.

[31] S. Fusi and M. Mattia, "Collective behavior of networks with linear (VLSI) integrate and fire neurons", *Neural Computation*, 1997.

[32] H. F. Yanai and Y. Sawada, "Integrator neurons for analogue neural networks", *IEEE Transactions on Circuits and Systems*, vol. 37, pp. 854–856, 1990.

[33] B. Kosko, *Neural Networks and Fuzzy Systems*, Prentice Hall International, 1992.

[34] J. A. Sutherland, "Holographic model of memory, learning and expression", *International Journal of Neural Systems*, vol. 1, pp. 256–267, 1990.

[35] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning", *Artificial Intelligence Review*, vol. 11, no. 1, pp. 11–73, 1997.

[36] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.

[37] V. Vapnik, "The support vector method of function estimation", in C. M. Bishop, editor, *Neural Networks and Machine Learning*, pp. 239–268. Springer-Verlag, 1998.

[38] C. J. C. Burges, "Support vector machines and kernel based methods", *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[39] M. Jordan, "Why the logistic function? A tutorial discussion on probabilities and neural networks", Technical Report 9503, Computational Cognitive Science, MIT, Cambridge, MA, 1995.

[40] J. A. Anderson, "Logistic discrimination", in *Handbook of statistics 2: Classification, pattern recognition and reduction of dimensionality*, pp. 169–191. North Holland, Amsterdam, 1982.

[41] W. Duch, R. Adamczak, and K. Grąbczewski, "Methodology of extraction, optimization and application of logical rules", in *Intelligent Information Systems VIII*, Ustroń, Poland, 1999. pp. 22–31.

[42] T. Poggio and F. Girosi, "A theory of networks for approximation and learning", Technical Report A. I. Memo 1140, MIT, Massachusetts, July 1989.

[43] F. Girosi and T. Poggio, "Networks and the best approximation property", AI Lab. Memo, MIT, 1989.

[44] D. R. Wilson and T. R. Martinez, "Value difference metrics for continuously valued attributes", in *Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, 1996, pp. 11–14.

[45] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions", *Journal of Artificial Intelligence Research*, vol. 6, pp. 1–34, 1997.

[46] D. R. Wilson and T. R. Martinez, "Heterogeneous radial basis function networks", in *Proceedings of the International Conference on Neural Networks*, June 1996, vol. 2, pp. 1263–1276.

[47] S. Ridella, S. Rovetta, and R. Zunino, "Circular backpropagation networks for classification", *IEEE Transaction on Neural Networks*, vol. 8, no. 1, pp. 84–97, 1997.

[48] N. N. Schraudolph, "A fast, compact approximation of the exponential function", Technical report, IDSIA, Lugano, Switzerland, 1998.

[49] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoid function", *IEEE Transactions on Information Theory*, vol. 39, pp. 930–945, 1993.

[50] V. Kurkova, "Approximation of functions by perceptron networks with bounded number of hidden units", *Neural Networks*, vol. 8, no. 5, pp. 745–750, 1995.

[51] V. Kurkova, P. C. Kainen, and V. Kreinovich, "Estimates of the number of hidden units and variation with respect to half-spaces", *Neural Networks*, vol. 10, no. 6, pp. 1061–1068, 1997.

[52] S. Qian, Y. C. Lee, R. D. Jones, C. W. Barnes, and K. Lee, "Function approximation with an orthogonal basis net", in *Proceedings of the IJCNN*, 1990, vol. 3, pp. 605–619.

[53] Mu-Song Chen, *Analyses and Design of Multi-Layer Perceptron Using Polynomial Basis Functions*, PhD thesis, The University of Texas at Arlington, 1991.

[54] K. Hornik, M. Stinchcombe, H. White, and P. Auer, "Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives", *Neural Computation*, vol. 6, no. 6, pp. 1262–1275, 1994.

[55] 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*, Oxford, 1987. Oxford University Press, pp. 143–167.

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

[57] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, San Diego, 1972.

[58] T. Poggio and F. Girosi, "Network for approximation and learning", *Proceedings of the IEEE*, vol. 78, pp. 1481–1497, 1990.

[59] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks", *Complex Systems*, vol. 2, pp. 321–355, 1988.

[60] D. Lowe, "Adaptive radial basis function nonlinearities, and the problem of generalization", in *1st IEE International Conference on Artificial Neural Networks*, London, UK, 1989. pp. 171–175.

[61] D. Lowe, "On the iterative inversion of RBF networks: A statistical interpretation", in *2nd IEE International Conference on Artificial Neural Networks*, London, UK, 1991. pp. 29–33.

[62] D. Lowe, "Novel "topographic" nonlinear", in *3rd IEE International Conference on Artificial Neural Networks*, London, UK, 1993. pp. 29–33.

[63] C. M. Bishop, "Improving the generalization properties of radial basis function neural networks", *Neural Computation*, vol. 3, no. 4, pp. 579–588, 1991.

[64] J. Allison, "Multiquadratic radial basis functions for representing multidimensional high energy physics data", *Computer Physics Communications*, vol. 77, pp. 377–395, 1993.

[65] W. Duch and G. H. F. Diercksen, "Feature space mapping as a universal adaptive system", *Computer Physics Communications*, vol. 87, pp. 341–371, 1994.

[66] A. Saranli and B. Baykal, "Complexity reduction in radial basis function (RBF) networks by using radial b-spline functions", *Neurocomputing*, vol. 18, no. 1-3, pp. 183–194, Feb. 1998.

[67] R. Durbin and D. E. Rumelhart, "Product units: A computationally powerful and biologically plausible extension to backpropagation networks", *Neural Computation*, vol. 1, no. 1, pp. 133–142, 1989.

[68] P. Niyogi and F. Girosi, "On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions", *Neural Computation*, vol. 8, no. 4, pp. 819–842, 1996.

[69] J. H. Friedman, "Multivariate adaptive regression splines (with discussion)", *Ann. Stat.*, vol. 19, pp. 1–141, 1991.

[70] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units", *Neural Computation*, vol. 1, no. 2, pp. 281–294, 1989.

[71] L. Bottou and V. Vapnik, "Local learning algorithms", *Neural Computation*, vol. 4, no. 6, pp. 888–900, 1992.

[72] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks", *Neural Computation*, vol. 5, no. 6, pp. 954–975, 1993.

[73] P. R. Krishnaiah and L. N. Kanal, *Handbook of statistics 2: Classification, pattern recognition and reduction of dimensionality*, North Holland, Amsterdam, 1982.

[74] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs with relationships to statistical pattern recognition", in F. Fogelman Soulié and J. Hérault, editors, *Neurocomputing: Algorithms, Architectures and Applications*, pp. 227–236. Springer-Verlag, New York, 1990.

[75] W. Duch, R. Adamczak, and G. H. F. Diercksen, "Distance-based multilayer perceptrons", in *International Conference on Computational Intelligence for Modelling Control and Automation*, Vienna, Austria, Feb. 1999. pp. 75–80.

[76] W. Duch, R. Adamczak, and G. H. F. Diercksen, "Neural networks in non-euclidean spaces", *Neural Processing Letters*, vol. 10, pp. 1–10, 1999.

[77] M. J. Kirby and R. Miranda, "Circular nodes in neural networks", *Neural Computations*, vol. 8, no. 2, pp. 390–402, 1996.

[78] W. Duch and N. Jankowski, "New neural transfer functions", *Journal of Applied Mathematics and Computer Science*, vol. 7, no. 3, pp. 639–658, 1997.

[79] R. Adamczak, W. Duch, and N. Jankowski, "New developments in the feature space mapping model", in *Third Conference on Neural Networks and Their Applications*, Kule, Poland, Oct. 1997. pp. 65–70.

[80] V. Kadirkamanathan, "A statistical inference based growth criterion for the RBF networks", in *Proceedings of the IEEE. Workshop on Neural Networks for Signal Processing*, 1994.

[81] N. Jankowski and V. Kadirkamanathan, "Statistical control of RBF-like networks for classification", in *7th International Conference on Artificial Neural Networks*, Lausanne, Switzerland, Oct. 1997. Springer-Verlag, pp. 385–390.

[82] N. Jankowski and V. Kadirkamanathan, "Statistical control of growing and pruning in RBF-like neural networks", in *Third Conference on Neural Networks and Their Applications*, Kule, Poland, Oct. 1997. pp. 663–670.

[83] W. Duch, R. Adamczak, and K. Grąbczewski, "Extraction of crisp logical rules using constrained backpropagation networks", in *International Conference on Artificial Neural Networks (ICANN'97)*, June 1997, pp. 2384–2389.

[84] W. Duch, R. Adamczak, and K. Grąbczewski, "Extraction of logical rules from backpropagation networks", *Neural Processing Letters*, vol. 7, pp. 1–9, 1998.

[85] N.J. Nillson, *Learning machines: Foundations of trainable pattern classifying systems*, McGraw-Hill, New York, 1965.

[86] B. Schölkopf, C. Burges, and A. Smola, *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA, 1998.

[87] W. Duch, K. Grudzinski, and G. H. F. Diercksen, "Minimal distance neural methods", in *World Congress of Computational Intelligence*, Anchorage, Alaska, May 1998. pp. 1299–1304.

[88] W. Duch, "Neural minimal distance methods", in *Proceedings 3-rd Conference on Neural Networks and Their Applications*, Kule, Poland, Oct. 1997. pp. 183–188.

[89] C. J. Merz and P. M. Murphy, "UCI repository of machine learning databases", 1998, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[90] W. Duch, N. Jankowski, A. Naud, and R. Adamczak, "Feature space mapping: a neurofuzzy network for system identification", in *Proceedings of the European Symposium on Artificial Neural Networks*, Helsinki, Aug. 1995. pp. 221–224.

[91] N. Jankowski, "Approximation with RBF-type neural networks using flexible local and semi-local transfer functions", in *4th Conference on Neural Networks and Their Applications*, May 1999, pp. 77–82.