ESANN'2001 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2001, D-Facto public., ISBN 2-930307-01-3, pp. 81-94

Transfer functions: hidden possibilities for better neural networks.

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

Abstract. Sigmoidal or radial transfer functions do not guarantee the best generalization nor fast learning of neural networks. Families of parameterized transfer functions provide flexible decision borders. Networks based on such transfer functions should be small and accurate. Several possibilities of using transfer functions of different types in neural models are discussed, including enhancement of input features, selection of functions from a fixed pool, optimization of parameters of general type of functions, regularization of large networks with heterogeneous nodes and constructive approaches. A new taxonomy of transfer functions is proposed, allowing for derivation of known and new functions by additive or multiplicative combination of activation and output functions.

1 Introduction

Two most popular feedforward neural networks models, the multi-layer perceptron (MLP) and the Radial Basis Function (RBF) networks, are based on specific architectures and transfer functions. MLPs use sigmoidal transfer functions, while RBFs use radial functions, usually Gaussians. Both types of networks are universal approximators [2]. This is an important, but almost trivial property, since networks using non-polynomial transfer functions are always universal approximators [20]. The speed of convergence and the complexity of networks needed to solve a given problem is more interesting. Approximations of complex decision borders or approximations of multidimensional mappings by neural networks require flexibility that may be provided only by networks with sufficiently large number of parameters. This leads to the bias-variance dilemma [2], since large number of parameters contribute to a large variance of neural models and small number of parameters increase their bias.

Regularization techniques may help to avoid overparameterization and reduce variance but training large networks has to be expensive. Moreover, in MLP models regularization methods decrease the weights forcing the network function to be more smooth. For some data where sharp, rectangular decision borders are needed, this may be a problem. Suppose that a simple logical rule:

$$\text{IF } x_1 > 0 \wedge x_2 > 0 \text{ THEN Class1 ELSE Class2}$$

has been used to create data samples. This distribution may easily be represented by two hyperplanes, but there is no way to represent it accurately with soft sigmoidal functions used in MLPs or Gaussian functions used by RBF networks. As a result for some datasets no change in the learning rule or network architecture will improve the accuracy of neural solutions. A real-world example is the hypothyroid dataset, for which the best optimized MLPs still give about 1.5% of error [24] while logical rules reduce it to 0.64% [9, 5]. Some real world examples showing the differences between RBF and MLP networks that are mainly due to the transfer functions used were presented in [12]. Artificial data for which networks using sigmoidal functions (hyperplanar, delocalized decision borders) need $O(N)$ parameters while networks with localized transfer functions (for example Gaussians) need $O(N^2)$ parameters, and artificial data in which the situation is reversed, are presented in [12].

Although all these networks are universal approximators in real applications their speed of learning and final network complexity may differ significantly. In principle neural networks may learn any mapping, but the inner ability to learn quickly in a given case may require flexible “brain modules”, or transfer functions that are appropriate for the problem to be solved. Learning and architectures, the main research topics in neural computing, will not solve this problem. There is “no free lunch" [14], i.e. no single learning algorithm is inherently superior to all the others. In real applications it may be better to find an appropriate model before embarking on a tedious task of selecting the best architecture and network training.

Selection and/or optimization of transfer functions performed by artificial neurons have been so far little explored ways to improve performance of neural networks in complex problems. This paper tries to clarify some of the issues connected with use of different transfer functions in neural networks. The next section discusses several ways in which transfer functions may be exploited. The third section contains a new taxonomy of these functions showing how to systematically generate functions useful for heterogeneous networks. Conclusions are given in the last section.

2 The use of transfer functions in neural models

Transfer functions may be used in the input pre-processing stage or as an integral part of the network. In the last case, transfer functions contain adaptive parameters that are optimized. The simplest approach is to test several networks with different transfer functions and select the best one. Using heterogenous functions in one network may give better effects. Starting from a network with several types of transfer function one may train it, possibly using pruning techniques to drop functions that are not useful. Constructive methods may also be used, training several candidate nodes and selecting the one that is the best performer. These possibilities are briefly discussed below.

Pre-processing of inputs. The hidden layer of a neural network maps the inputs into an image space trying to simplify the task of the perceptron output node, for example by creating linearly separable data clusters. Instead of the hidden layer transfer functions that contain some adaptive parameters one could use arbitrary multivariate functions to transform inputs trying to achieve similar result.

In the *functional link* (FL) networks of Pao [22] combination of various functions, such as polynomial, periodic, sigmoidal and Gaussian functions is used. These networks were never popular and little is known about their properties. The use of products of pairs $x_i x_j$ or of higher order products is not very practical for high-dimensional inputs because the number of such products grows quickly. Functional link networks have an extra layer with nodes that compute products, periodic functions and other functions, and that pass the results as inputs to an MLP. For example, in the functional model Pao recommends orthogonal basis functions. These pre-processing functions should be regarded rather as filter functions than transfer functions.

Pre-processing may be done independently of the network by basis functions $\Phi(\mathbf{x})$ (acting on the whole input vector $\mathbf{x}$ or on a few elements only) if they do not involve adaptive parameters. Weighted combination of enhanced inputs is always performed by the network. However, filter functions that have adaptive parameters should be a part of the network. To avoid excessive number of inputs one could form a candidate input node and evaluate its information content using feature-evaluation techniques [16] before adding new dimension to the input space.

Except for adding filtered inputs to the existing inputs one may renormalize all input vectors, for example using Minkovsky's metric. Such input renormalization has dramatic influence on network decision borders [6]. Adding a new feature based on sum of squares $\Phi(\mathbf{x}) = \sqrt{||\mathbf{x}||^2_{\max} - \sum_i x_i^2}$ creates circular contours of constant value $\Phi(\mathbf{x})$=const and renormalizes all enhanced vectors to $||(\mathbf{x}, \Phi(\mathbf{x}))|| = ||\mathbf{x}||_{\max}$. If renormalization using Minkovsky's metric is desired then $\Phi(\mathbf{x})^{\alpha} = ||\mathbf{x}||^{\alpha}_{\max} - \sum_i |x_i|^{\alpha}$, where $||\mathbf{x}||$ is now the Minkovsky's norm.

Selection of functions for homogenous networks. In this case all functions in neural network are of the same type, selected from a fixed pool of different transfer functions. For example, several radial functions may be tried in an RBF network and functions that perform in the best way selected. Other type of functions, such as separable transfer functions, may be used in similar architectures [10, 8, 18]. A survey of many known and some new functions suitable for neural modeling has been published recently [12]. A systematic taxonomy of these functions has also been presented [13]. Networks based on rational functions, various spline functions, tensor products, circular units, conical, ellipsoidal, Lorentzian functions and many others were described in the literature [12].

Unfortunately there are no neural simulators that would allow for easy evaluation and selection of networks with transfer functions of many different types, although some RBF, FSM or IncNet networks may be used to test a few functions. In the distance-based MLPs (D-MLP) a whole class of transfer functions is introduced [7, 6] by changing the weighted activation in the sigmoidal transfer functions $\sigma(\mathbf{w} \cdot \mathbf{x} - \theta) = \sigma(d_0 - D(\mathbf{w}, \mathbf{x}))$ into a distance function; in particular if $D(\mathbf{w}, \mathbf{x})$ is the square of Euclidean distance the same type of hyperplanar transfer functions as in the standard MLP are obtained. Using the Minkovsky's distance function $D_{\alpha}(\mathbf{w}, \mathbf{x})$ taken to the $^{\beta}$ power, a family of neural networks with transfer functions parameterized by α, β is obtained. The standard MLP with hyperplanar contours of single neurons corresponds to $\alpha = \beta = 2$; for other values of parameters quite different decision borders are provided by these transfer functions. Several families of transfer functions may be parameterized allowing for selection of networks based on functions that are most natural for a given data.

Support Vector Machines [3] are used with different kernel functions. Both neural networks and SVMs are wide margin classifiers. At least part of the SVM success may be due to the selection of the best kernel for a given data, although for simple benchmark problems the differences between results obtained with different kernels (or different transfer functions) may be small.

Heterogenous function networks. Mixed transfer functions within one network may be introduced in two ways. A constructive method that selects the most promising function from a pool of candidates in RBF-like architecture and adds it to the network has been introduced [8, 19]. Other constructive algorithms, such as the cascade correlation [15], may also be used for this purpose. Each candidate node using different transfer function should be trained and the most useful candidate added to the network.

In the second approach starting from a network that already contains several types of functions, such as Gaussian and sigmoidal functions, pruning or regularization techniques are used to reduce the number of functions [19]. Initially the network may be too complex but at the end only the functions that are best suited to solve the problem are left. In the ontogenic approach neurons are removed and added during the learning process [19].

Very little experience with optimization of transfer functions has been gathered. Neural networks using different transfer functions should use less nodes and thus the function performed by the network may be more transparent. For example, one hyperplane may be used to divide the input space into two classes and one additional Gaussian function to account for local anomaly. Analysis of the mapping performed by an MLP network trained on the same data will not be so simple.

3 Transfer functions and their parameterization

Transfer functions should provide maximum flexibility of their contours with small number of adaptive parameters. Large networks with simple neurons may have the same power as small networks with more complex neurons. Recently a detailed survey of transfer functions has been published [12], containing all relevant references. Here a new taxonomy of these functions, based on their flexibility, is proposed, starting from the simplest functions and building more flexible functions with reasonable number of adaptive parameters.

Two functions determine the way signals are processed by neurons. *The activation function* acting on the input vector $I(\mathbf{x})$ determines the total signal a neuron receives, and *the output function* $o(I)$, operating on scalar activation, determines the scalar output. The composition of the activation and the output function is called the *transfer function* $o(I(\mathbf{x}))$. For some transfer functions there is no natural division between activation and output functions.

A fan-in function, i.e. weighted combination of the incoming signals, is the most common activation. For neuron i connected to neurons j (for $j = 1,\ldots,N$) sending signals x_j with the strength of the connections w_j the total activation $I(\mathbf{x};\mathbf{w})$ is

$$I(\mathbf{x};\mathbf{w}) = \sum_{j=0}^{N} w_j x_j \tag{1}$$

where $w_0 = \theta$ (threshold) and $x_0 = 1$. This activation has biological inspirations and leads to hyperplanes as contours of $I(\mathbf{x}) = const$. 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. 3 main choices for activation functions are:

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

The *output function* $o(I)$ operates on scalar activations and returns scalar values. Typically a squashing function is used to keep the output values within specified bounds. The 3 major choices here are: 1. sigmoidal non-local functions; 2. functions localized around a single center; 3. semi-centralized functions that have either many centers or hard to define centers.

Transfer functions are divided below into 4 categories summarized in Table 1, starting with the simplest step-like functions, their smooth generalization in form of radial and sigmoidal functions, moving to multivariate functions with independent parameters for each dimension and finally reaching the level of the most flexible transfer functions.

3.1 Step–like functions

Fan-in activation used here leads to hyperplanar contours of transfer functions. Output function of logical neurons is of *the step function* type, known as the Heaviside $\Theta(I;\theta)$ function:

$$\Theta(I;\theta) = 1 \text{ if } I > \theta \text{ and } 0 \text{ if } I \leq \theta \tag{2}$$

The greatest advantage of using logical elements is the high speed of computations and easy realization in the hardware. Networks of such logical neurons divide the input space into polyhedral areas.

Multi-step functions allow to realize multi-valued logic. These functions have a number of thresholds:

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

With fan-in activation they define a series of parallel hyperplanes. Semi-linear functions:

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

allow to define hyperplanar contours for any $\theta_1 < I \leq \theta_2$. Step-like functions with distance-based activations are also worth considering although they probably have never been used.

3.2 Sigmoidal transfer functions

Step-like functions have discontinuous derivatives, preventing the use of gradient-based error minimization training procedures. *Sigmoidal output* functions allow to define popular continuous *graded response neurons*, for example the logistic output function:

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

The constant s determines the slope of the logistic function. There are many functions similar in σ–shape to the logistic function, forming a broad class of *sigmoidal functions*. Combined with the fan-in functions they give non-local transfer functions. Combined with other activations (see below, Eq. 12) sigmoidal functions may produce localized transfer functions. Logistic functions may be replaced by the error (erf) function, arcus tangent or the hyperbolic tangent functions [12], but since calculation of exponents is much slower than arithmetic operations other functions of sigmoidal shape may be useful to speed up computations. For example:

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

$$s_3(I;s) = \frac{sI}{1+|sI|} \tag{7}$$

3.3 Radial basis functions and their approximations

Radial basis functions (RBFs) are used in approximation theory and in pattern recognition [17]. They use the radial coordinate $r = ||\mathbf{x}-\mathbf{t}||$ as an activation; this coordinate also forms the simplest RBF function

$$h(r) = r = ||\mathbf{x}-\mathbf{t}|| = D(\mathbf{x};\mathbf{t}). \tag{8}$$

General multiquadratic functions, thin-plate spline functions

$$h_1(r,b) = (b^2+r^2)^{-\alpha} \tag{9}$$

$$h_3(r,b) = (br)^2\ln(br) \tag{10}$$

and many other radial functions are non-local. Among localized radial basis functions *Gaussian functions*

$$G(r,b) = e^{-r^2/b^2} \tag{11}$$

are unique since for Euclidean (and some other) distance functions they are separable, i.e. equal to products of independent factors for each of the input components. Logistic function, tanh or simple inverse quadratic and quartic functions approximate roughly the shape of a Gaussian function:

$$G_1(r) = 2 - 2\sigma(r^2); \qquad G_2(r) = 1 - \tanh(r^2); \qquad G_{2n}(r) = \frac{1}{1+r^{2n}} \tag{12}$$

Radial cubic and quadratic B-spline function were also used to approximate Gaussian function [12].

3.4 Multivariate functions

The multivariate Gaussian functions give localized hyperellipsoidal output contours:

$$G_g(\mathbf{x};\mathbf{t},\mathbf{b}) = e^{-D^2(\mathbf{x};\mathbf{t},\mathbf{b})} = \prod_{i=1}^{N} e^{-(x_i-t_i)^2/b_i^2} \tag{13}$$

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

$$G_S(\mathbf{x};\mathbf{t},\mathbf{b}) = 1 - \sigma(D(\mathbf{x};\mathbf{t},\mathbf{b})^2) = \frac{1}{1+e^{D^2(\mathbf{x};\mathbf{t},\mathbf{b})}} \tag{14}$$

For N-dimensional input space each ellipsoidal unit uses $2N$ adaptive parameters. A single unit may also provide more complex contours if more general distance functions are used:

$$\bar{G}_2(\mathbf{x};\mathbf{t},\mathbf{b}) = \prod_{i=1}^{N} \frac{1}{1+(x_i-t_i)^2/b_i^2} \tag{15}$$

or with:

$$\bar{G}_3(\mathbf{x};\mathbf{t},\mathbf{b}) = \frac{1}{1+\sum_{i=1}^{N}(x_i-t_i)^2/b_i^2} = \frac{1}{1+D^2(\mathbf{x};\mathbf{t},\mathbf{b})} \tag{16}$$

Large dispersions b_i make Gaussian factors for inputs x_i almost constants within their variability range, allowing to remove the irrelevant dimensions. Instead of multidimensional Gaussian functions a combination of one-dimensional Gaussians may be used:

$$\bar{G}(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{v}) = \sum_{i=1}^{N} v_i e^{-(x_i-t_i)^2/b_i^2} \tag{17}$$

The advantage of this additive “Gaussian bar" functions is that it make elimination of irrelevant input variables easier than in the multiplicative multidimensional Gaussian case. Combination of sigmoidal functions creates *sigmoidal bar* function:

$$\bar{\sigma}(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{v}) = \sum_{i=1}^{N} \frac{v_i}{1+e^{-(x_i-t_i)^2/b_i^2}} \tag{18}$$

The *Lorentzian* response functions use the squared fan-in activation function to create contours of non-localized window-type:

$$L(\mathbf{x};\mathbf{w}) = \frac{1}{1+I^2(\mathbf{x};\mathbf{w})} = \frac{1}{1+\left(\sum_{i=1}^{N} w_i x_i - \theta\right)^2} \tag{19}$$

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 other transfer functions, for example:

$$W(\mathbf{x};\mathbf{w}) = e^{I(\mathbf{x};\mathbf{w})^2} \tag{20}$$

3.5 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 values of parameters are localized, and for other values non-localized. Ridella *et al.* [23] used circular units in their Circular Backpropagation Networks. The output function is a standard sigmoid while the activation function contains one extra term:

$$A_R(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{N} w_i x_i + w_{N+1} \sum_{i=1}^{N} x_i^2 \tag{21}$$

Dorffner [4] 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:

$$A_C(\mathbf{x};\mathbf{w},\mathbf{t},\omega) = I(\mathbf{x}-\mathbf{t};\mathbf{w}) + \omega D(\mathbf{x}-\mathbf{t}) \tag{22}$$

Many other combinations of fan-in and distance functions could also serve as universal transfer functions. For example, $\exp(\alpha I^2 - \beta D^2)$ function. The approximated Gaussian combined with the Lorentzian function also provides an interesting universal transfer function:

$$C_{GL1}(\mathbf{x};\mathbf{w},\mathbf{t},\alpha,\beta) = \frac{1}{1+(\alpha I(\mathbf{x};\mathbf{w}) + \beta D(\mathbf{x};\mathbf{t}))^2} \tag{23}$$

or

$$C_{GL2}(\mathbf{x};\mathbf{w},\mathbf{t},\alpha,\beta) = \frac{1}{1+\alpha I^2(\mathbf{x};\mathbf{w}) + \beta D^2(\mathbf{x};\mathbf{t})} \tag{24}$$

Bicentral family of transfer functions is formed from N pairs of sigmoids:

$$Bi(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) = \prod_{i=1}^{N} \sigma(A1_i^+)\ (1-\sigma(A1_i^-)) \tag{25}$$

where $A1_i^\pm = s_i \cdot (x_i - t_i \pm b_i)$ and $\sigma(x)$ is a logistic function (Eq. 5). 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 $||x-t||$, while here components of two centers, t_i+b_i and t_i-b_i, are used, therefore these functions are called *bicentral*. They use only $3N$ adaptive parameters, are localized (window-type) and separable. In contrast to the Gaussian functions a single bicentral function may approximate rather complex decision borders.

The next step towards even greater flexibility requires rotation of contours provided by each unit [11, 1]. Full $N \times N$ rotation matrix is very hard to parameterize with $N-1$ independent rotation angles. Using transfer functions with just $N-1$ additional parameters per neuron one can implement rotations defining a product form of the combination of sigmoids:

$$C_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{r}) = \prod_{i}^{N} \sigma(A3_i^+)(1-\sigma(A3_i^-)) \tag{26}$$

where $A3_i^\pm = s_i(x_i + r_i x_{i+1} - t_i \pm b_i)$ and $x_{N+1} = x_1$. Another solution to this problem is presented in [12].

3.6 Generalized universal transfer functions

Functions suitable for neural models should form families parametrerized for greatest flexibility. An extension of the conic activation (22) is defined by:

$$A_C(\mathbf{x};\mathbf{w},\mathbf{t},\mathbf{b},\alpha,\beta) = \alpha I(\mathbf{x}-\mathbf{t};\mathbf{w}) + \beta D(\mathbf{x};\mathbf{t},\mathbf{b}) \tag{27}$$

Used with the sigmoidal output function this generalized conical function changes smoothly from a localized Gaussian (with independent biases) to a soft hyperplane type. Such functions may be used in heterogenous neural networks [19].

Adding rotations to Ridella function gives another very flexible function:

$$A_{GR}(\mathbf{x};\mathbf{w},\mathbf{s},\mathbf{r}) = w_0 + \sum_{i=1}^{N} \left(w_i x_i + s_i (x_i + r_i x_{i+1})^2 \right) \tag{28}$$

where $x^r = [x_2, \ldots, x_{N-1}, x_1]$ and $x_{N+1} = x_1$.

Bicentral functions with two independent slopes increase contour flexibility:

$$Bi2s(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) = \prod_{i=1}^{N} \sigma(A2_i^+)\ (1-\sigma(A2_i^-)) \tag{29}$$

where $A2_i^\pm = s_i^\pm (x_i - t_i \pm b_i)$. Using small slope s_i^+ and/or s_i^- this generalized bicentral function may delocalize or stretch to *left* and/or *right* in any dimension and become semi–localized. This enables half-infinite channel, half-hyperellipsoidal, soft triangular and many other contour shapes. Each function requires $4N$ parameters.

The most flexible bicentral function is obtained by combining rotations with two independent slopes (Fig. 1):

$$BiR2s(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s},\mathbf{r}) = \prod_{i=1}^{N} \sigma(A4_i^{+})(1-\sigma(A4_i^{-})) \tag{30}$$

where $A4_i^{\pm} = s_i^{\pm}(x_i + r_i x_{i+1} - t_i \pm b_i)$, and $r_1, \ldots, r_{N-1}$ define the rotation; $x_{N+1} = 0$ 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.

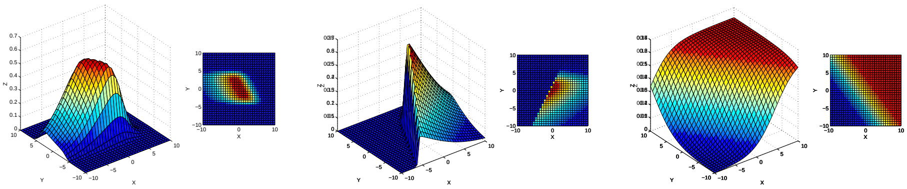

Figure 1: A single bicentral transfer functions has very flexible contours.

Transfer functions are presented in hierarchical order according to their flexibility in Table 1. The most general families of functions are placed at the top row of the Table, and the simplest functions are in the bottom row. Activation and output function type is mentioned for each transfer function. I designates the fun-in activation (1), I^{+} transformed input or weight vector used in this activation and $I_i = x_i w_i$. D is the Euclidean distance; $D^b = \sqrt{\sum((x_i - t_i)/b_i)^2}$, and $D_i = (x_i - t_i)/b_i$. G is the Gaussian, σ is the logistic function, $\prod$ and $\sum$ are used to describe *products of* or *sum of* terms. A in description of output referes to the activation function or its component.

Various activations I and D, together with localized Gaussian and delocalized sigmoidal factors, combined in additive or multiplicative way allows for construction of all transfer functions. For example, the logistic function comes from combination of activation I and output σ, while the Gaussian transfer function from D and G. Combining these two activation functions $I + D$, and adding output σ, gives the conic transfer function. Thus the Table allows for systematic construction of transfer functions that may be used in neural models.

4 Conclusions

According to the Statlog report [21] comparing many classification methods on 22 real-world datasets, results of RBF and MLP differ significantly. In some cases cross-validation errors were twice as large using MLP than RBF while for other data the situation was reversed. Although one may always argue that better initialization, learning

Bicentral (2Slope, Rot2Slope, . . .) (29, 30, [12]) Act: $A2$–$A4$, O: $\prod(Ai^-, Ai^+, \sigma)$	**G-Conic** (27) Act: $I + D^b$, O: σ	**G-Ridella** (28) Act: $I^+ + D^+$, O: σ		
Bicentral (25,26) Act: $A1, A3$, O: $\prod(Ai^-, Ai^+, \sigma)$	**Conic** (22) Act: $I + D$, O: σ	**Ridella** (21) Act: $I^+ + D^+$, O: σ	C_{GL1} (23) Act: $I + D$, O: $\frac{1}{1+A}$	C_{GL1} (23) Act: $I + D$, O: $\frac{1}{1+A}$
Multivariate Gaussian (13) Act: D^b, O: G	**Multivariate Sigmoid** (14) Act: D^b, O: σ	$\bar{G}_2$ (15) Act: D_i, O: $\prod \frac{1}{1+A}$	$\bar{G}_3$ (16) Act: D_i, O: $\frac{1}{1+\sum A}$	
Gaussian-bar (17) Act: D^b, O: $\sum G$	**Sigmoidal-bar** (18) Act: D^b, O: $\sum \sigma$	**Lorentzian** (19) Act: I, O: $\frac{1}{1+\sum A}$	**Window** (20) Act: I, O: G	
Gaussian (11) Act: D, O: G	**Radial coordinate** (8) Act: D, O: A	**Multiquadratics** (9) Act: D, O: $(b^2 + D^2)^\alpha$	**Thin-plate spline** (10) Act: D, O: $(bD)^2 \ln(bD)$	
Gaussian Approximations (12) Act: D, O: $G_1 = 2 - 2\sigma(r^2)$, $G_2 = \tanh(r^2)$, $G_{2n} = \frac{1}{1+r^{2n}}$, splines approx. [12]				
Logistic (5) Act: I, O: σ	**Other Sigmoids** Act: I, O: tanh, arctan	**Sigmoids Approximations ($s2, s3$)** (6–7) Act: I, O: $\Theta(I)\frac{I}{I+s} - \Theta(-I)\frac{I}{I-s}$, $\frac{sI}{1+\sqrt{1+s^2I^2}}$, $\frac{sI}{1+\|sI\|}$, $\frac{sI}{\sqrt{1+s^2I^2}}$		
Heaviside (2) Act: I, O: $\Theta(I;\theta)$	**Multistep** (3) Act: I, O: $\varsigma(I)$	**Semi-linear** (4) Act: I, O: $s_l(I;\theta_1,\theta_2)$		

Table 1: Transfer functions hierarchically ordered according to their flexibility. Each gray row contains functions of similar flexibility. Top rows contain the most flexible functions and bottom rows the simplest. Names of functions and types of activation/output function functions are given. For detailed description see text.

and architectures will reduce the difference, at least part of that difference may be attributed to different shapes of decision borders (hyperplane vs. ellipsoidal) provided by their transfer functions.

Most neural networks are based on either sigmoidal or Gaussian transfer functions. We have described several possibilities for introduction of different transfer functions: selecting functions from a fixed set, introduction of parameterized families of functions that may be optimized in networks using the same type of nodes (homogenous networks) or in networks using different nodes (heterogenous networks), or using constructive algorithms that add and optimize new functions when the data requires more complex description. Flexibility of a single processing unit, increasing with the number of adjustable parameters per function, should be balanced with 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. The optimal balance between the number and the complexity of units may strongly depend on problems to be solved.

Although for simple problems it may not matter in some applications selection of transfer functions may make a difference. When the datasets are small and the dimensionality of the feature space relatively large the actual shape of decision borders is irrelevant and even logical rules or decision trees, dividing the feature space into hyperboxes, may be sufficient. The more data samples are provided the more inadequacy of the models will show up, and the need for flexible contours approximating real distribution of data will become apparent. We have presented here a taxonomy of functions suitable for neural network models, starting from the simplest step-like functions and proceeding to the most flexible functions judiciously parameterized. For mutidimensional problems these functions should be much more attractive than polynomials, improving upon convergence of sigmoidal and Gaussian functions.

Investigation of the role of transfer functions has just started and little is known about merits of some 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. Very few experiments with heterogeneous transfer functions in single networks have been made. Training algorithms have developed to the point where returns may be diminishing, while investigation of the neural transfer functions still hide unexplored possibilities.

Acknowledgments: Support by the Polish Committee for Scientific Research, grant 8 T11C 006 19, is gratefully acknowledged.

References

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

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

ESANN'2001 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2001, D-Facto public., ISBN 2-930307-01-3, pp. 81-94

[3] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines (and other kernel-based learning methods)*. Cambridge University Press, 2000.

[4] G. Dorffner. A unified framework for MLPs and RBFNs: Introducing conic section function networks. *Cybernetics and Systems*, 25(4):511–554, 1994.

[5] W. Duch, Adamczak, and K. Grąbczewski. Methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks*, March 2001.

[6] W. Duch, R. Adamczak, and G. H. F. Diercksen. Neural networks in non-euclidean spaces. *Neural Processing Letters*, 10:201–210, 1999.

[7] W. Duch, R. Adamczak, and G.H.F. Diercksen. Distance-based multilayer perceptrons. In M. Mohammadian, editor, *Computational Intelligence for Modelling Control and Automation. Neural Networks and Advanced Control Strategies*, pages 75–80. Amsterdam, 1999.

[8] W. Duch, R. Adamczak, and G.H.F. Diercksen. Constructive density estimation network based on several different separable transfer functions. In *9th European Symposium on Artificial Neural Networks*, Bruges, Belgium, April 2001.

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

[10] W. Duch and G. H. F. Diercksen. Feature space mapping as a universal adaptive system. *Computer Physics Communications*, 87:341–371, 1995.

[11] W. Duch and N. Jankowski. New neural transfer functions. *Journal of Applied Mathematics and Computer Science*, 7(3):639–658, 1997.

[12] W. Duch and N. Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2:163–212, 1999.

[13] W. Duch and N. Jankowski. Taxonomy of neural transfer functions. In Shun-Ichi Amari, C. Lee Giles, Marco Gori, and Vincenzo Piuri, editors, *International Join Conference on Neural Networks*, volume III, pages 477–484, Como, Italy & Los Alamitos, California, July 2000. Computer Society and IEEE.

[14] R. O. Duda, P. E. Hart, and D.G. Stork. *Patter Classification*. John Wiley and Sons, New York, 2001.

[15] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, 1990.

[16] M. Fernández and C. Hernández. Neural networks input selection by using the training set. In *IEEE World Congress on Computational Intelligence. Proceeding of the 1999 IEEE International Joint Conference on Neural Networks*, number 0411, Washington D.C., USA, 1999.

ESANN'2001 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2001, D-Facto public., ISBN 2-930307-01-3, pp. 81-94

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

[18] 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*, pages 77–82, Zakopane, Poland, May 1999.

[19] N. Jankowski and W. Duch. Universal neural networks. In *9th European Symposium on Artificial Neural Networks*, Bruges, Belgium, April 2001.

[20] M. Leshno, V.Y. Lin, Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, Neural Networks:861–867, 1993.

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

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

[23] S. Ridella, S. Rovetta, and R. Zunino. Circular backpropagation networks for classification. *IEEE Transaction on Neural Networks*, 8(1):84–97, 1997.

[24] W. Schiffman, M. Joost, and R. Werner. Comparison of optimized backpropagation algorithms. In *Proceedings of ESANN'93*, pages 97–104, Brussels, 1993.