Neural Networks from Similarity Based Perspective

Włodzisław DUCH, Rafał ADAMCZAK
Department of Computer Methods, Nicholas Copernicus University,
Grudziądzka 5, 87-100 Toruń, Poland; e-mail: duch,raad@phys.uni.torun.pl

and Geerd H.F. DIERCKSEN
Max-Planck Institute of Astrophysics, 85740-Garching, Germany,
e-mail: GDiercksen@mpa-garching.mpg.de

Abstract: A framework for Similarity-Based Methods (SBMs) includes many neural network models as special cases. Multilayer Perceptrons (MLPs) use scalar products to compute weighted activation of neurons, combining soft hyperplanes to provide decision borders. Scalar product is replaced by a distance function between the inputs and the weights, offering a natural generalization of the standard MLP model to the distance-based multilayer perceptron (D-MLP) model. D-MLPs evaluate similarity of inputs to weights making the interpretation of their mappings easier. Cluster-based initialization procedure determining architecture and values of all adaptive parameters is described. D-MLP networks are useful not only for classification and approximation, but also as associative memories, in problems requiring pattern completion, offering an efficient way to deal with missing values. Non-Euclidean distance functions may also be introduced by normalization of the input vectors in an extended feature space. Both approaches influence the shapes of decision borders dramatically. An illustrative example showing these changes is provided.

1 Introduction

Multilayer perceptrons (MLPs) trained with backpropagation method (BP) are certainly the most popular among all neural techniques [1]. Applied to classification or approximation problems MLPs use sigmoidal functions to provide soft hyperplanes dividing the input space into separate regions. MLPs are therefore similar to the statistical discriminant techniques or the Support Vector Machines (SVM) [2], although combination of soft sigmoids allows for representation of more complex, nonlinear decision borders. This is usually considered to be a strength of the MLP model, although in cases when sharp decision borders are needed it may also become its weakness. For example, classification borders conforming to a simple logical rule $x_1 > 1 \wedge x_2 > 1$ are easily represented by two hyperplanes but there is no way to represent them accurately using soft sigmoidal functions used in MLPs. This problem is especially evident if regularization terms are added to the cost function, enforcing small values of weights. As a result for some datasets no change in the learning rule or network architecture will improve the accuracy of neural solutions. A good real-world example is the hypothyroid dataset, for which the best optimized MLPs still give about 1.5% of error [3] while logical rules reduce it to 0.64% [4]. Most research on neural networks is concentrated on architectures and learning rules, but the selection of neural transfer functions may be crucial to network performance [5].

Another problem with MLP models concerns selection of architecture and initialization of adaptive parameters. Constructive neural algorithms [6] may help to find architectures that roughly match complexity of the data analyzed, but constructive models may also end up in suboptimal architectures. Genetic algorithms applied to selection of architectures do not guarantee good solutions and are computationally very demanding [7]. Missing inputs are especially difficult to handle since filling the unknown features with the most frequently appearing values may lead to poor results.

MLPs are widely used for classification and approximation problems, while many interesting problems involve pattern completion and association. Associative memory models are usually based on recurrent networks. It would be very interesting to accomplish similar task using feedforward MLP networks. MLPs, SVMs [2] and other methods based on discriminant analysis, perform mappings that are rather difficult to interpret. Proponents of the logical rule-based machine learning methods consider it to be the biggest drawback of neural networks, limiting their applications in safety-critical fields such as medicine. Similarity-Based Methods (SBMs), for example the k-nearest neighbor (k-NN) method, retrieve the relevant context for each query presented to the classification system, providing some interpretation and estimating probability of different class assignment. Such interpretation is also possible for the Radial Basis Function (RBF) networks using Gaussian or other localized functions, or the Learning Vector Quantization (LVQ) method based on optimization of reference vectors. It may seem that such an interpretation is not possible for MLPs since they belong to the discriminant rather than to memory-based techniques. One way to obtain an interpretation of MLP decisions is to study the transition from MLPs to networks performing logical operations [8]. Although discriminant methods and prototype methods seem to be quite different in fact the two approaches are deeply connected. A single hyperplane discriminating vectors belonging to two classes may be replaced by two prototypes, one for each class. For N prototypes one can generate $N(N-1)/2$ pair-wise discriminating hyperplanes providing piece-wise linear approximation to the decision borders.

All these shortcomings of the MLP networks are overcome here. Recently a general framework for Similarity-Based Methods (SBMs) used for classification has been presented [9]. It is briefly presented in the next section, and several examples of well-known and new neural methods derived using this framework are presented. In particular the Distance-Based Multilayer Perceptrons (D-MLPs) are introduced, improving upon the traditional approach by providing more flexible decision borders, using information about the structure of the data derived from clusterization procedures and enabling a prototype-based interpretation of the results. Symbolic values used with probabilistic distance functions allow to avoid ad hoc procedure to replace them with numerical values. SBM perspective allows to initialize all D-MLP network parameters starting from some one of standard clusterization procedures and thus using information that may be easily obtained from the data. A simple procedure to change D-MLP models into associative memories and to use them in pattern completion problems is described in the fourth section. As a result missing values are handled in an efficient way. Finally to avoid writing computer programs for the backpropagation method for each type of distance function a simple transformation of the input data is proposed, allowing for distance-based interpretation. An illustration of this method on the Iris data is presented in the sixth section. The paper is finished with a short discussion.

2 Neural methods from similarity-based perspective

The classification problem (the same reasoning may also be applied to regression and pattern completion problems) is stated as follows: given a set of N_t class-labeled training vectors $\{\mathbf{R}^j, \mathbf{C}(\mathbf{R}^j)\}$, $j = 1..N_t$, where $\mathbf{C}(\mathbf{R}^j)$ is the class of $\mathbf{R}^j$, and given a vector $\mathbf{X}$ of an unknown class, use the information provided in the similarity measure $D(\mathbf{X}, \mathbf{R}^j)$ to estimate the probability of classification $p(C_i|\mathbf{X}; M)$, where M describes the classification model used (values of all parameters and procedures employed). A general similarity-based model of an adaptive system used for classification should include at least the following elements:

$M = \{\{\mathbf{R}^j\}, D(\cdot), G(D(\cdot)), k, E[\cdot], K(\cdot), \mathcal{R}(\cdot|\cdot)\}$, where

$\{\mathbf{R}^j\}$ is the set of reference vectors created from the set of training vectors $\{\mathbf{X}^i\}$ by some procedure; $D(\cdot)$ is a similarity function (frequently a distance function) parameterized in various ways, or a table used to compute similarities; $G(D(\mathbf{X},\mathbf{R}))$ is a weighting function estimating contribution of the reference vector $\mathbf{R}$ to the classification probability; k is the number of reference vectors taken into account in the neighborhood of $\mathbf{X}$; $E[\cdot]$ is the total cost function optimized during training; it may include regularization terms and may depend on a kernel function $K(\cdot)$, scaling the influence of the error, for a given training example, on the total cost function, using a risk matrix $\mathcal{R}(C_i|C_j)$ that estimate the costs of assigning wrong classes.

The cost function that minimizes risk for overall classification is:

$$E(\{\mathbf{X}\};\mathcal{R},M) = \sum_i \sum_{\mathbf{X}} \mathcal{R}(C_i, C(\mathbf{X})) H(p(C_i|\mathbf{X};M), \delta(C_i, C(\mathbf{X}))) \tag{1}$$

where $i = 1 \dots N_c$ runs over all classes and $\mathbf{X}$ over all training vectors, $C(\mathbf{X})$ is the true class of the vector $\mathbf{X}$ and function $H(\cdot)$ is monotonic and positive, often a quadratic function. The elements of the risk matrix $\mathcal{R}(C_i,C_j)$ are proportional to the risk of assigning the C_i class when the true class is C_j, and in the simplest case $\mathcal{R}(C_i,C_j) = 2 - \delta_{ij}$ or $\mathcal{R}(C_i,C_j) = 1 + |i-j|$ is taken (strictly speaking a unit matrix is added here to the usual risk matrix). M specifies all adaptive parameters and variable procedures of the classification model that may affect the cost function. Regularization terms aimed at minimization of the complexity of the classification model are frequently added to the cost function, helping to avoid the overfitting problems. If $H(\cdot)$ is a quadratic function of the $\max_i p(C_i|\mathbf{X};M) - \delta(C_i, C(\mathbf{X}))$ standard mean square error (MSE) function is recovered.

An adaptive system may include several such models M_l and an interpolation procedure to select between different models or average results of a committee of models. Such averaging with boosting procedures for selection of training vectors leads to creation of stable and accurate classifiers [10]. Simple averaging, or linear combination of several models is most frequently used:

$$P(C_i|X;M) = \sum_{l=1}^{N} W_l \; p(C_i|X;M_l) \tag{2}$$

Least square minimization (LSM) procedure is used to determine W_l coefficients. Creating ensembles one should use all information available. Since we know for which training vectors R_k each model makes an error it seems reasonable to use this information in making an ensemble. Coefficients of linear combination should depend on the distance between X and those regions $R_{l,k}$ of the feature space where model M_l works poorly, therefore:

$$P(C_i|X;M) = \sum_{l=1}^{N} \sum_k W_l D(X, R_{l,k}) p(C_i|X;M_l) \tag{3}$$

should be a good choice. Identical LMS optimization is used as in the previous case. Probabilities are obtained after renormalization:

$$p(C_i|X;M) = P(C_i|X;M) / \sum_j P(C_j|X;M) \tag{4}$$

Many pattern recognition, machine learning and neural network models are a special case of this SBM framework. One way to use it is to start with the simplest model and turn on different optimization parameters and procedures, for example starting from the simples k-NN and optimizing the number of neighbors, distance function parameters, soft weighting, feature selection, number and position of reference vectors. Each step towards more complex model decreases the bias of the classifier, but may increase its variance [10], therefore after each step the model should be validated and only if the greater complexity is justified by higher accuracy more complex models should be accepted, otherwise a different type of optimization should be used.

2.1 RBF and LVQ-like methods

In RBF networks Euclidean distance functions $D(\mathbf{X},\mathbf{R}^j) = ||\mathbf{X}-\mathbf{R}^j||$ are assumed and a radial, for example Gaussian $G(D) = \exp(-D^2)$ weighting functions are used. Essentially RBF is a minimal distance soft weighted method with no restrictions on the number of neighbors – reference vectors $\mathbf{R}^j$ that are near influence probabilities of classification more than those that are far. The SBM framework suggests that there is nothing special about this choice of distance function and the weighting function. The simplest suitable weighting function is **the conical radial function**: zero outside the radius σ and $1-D(\mathbf{X},\mathbf{R})/\sigma$ inside this radius. Classification probability is calculated by the output node using the formula:

$$p(C_i|X;\sigma) = \frac{\sum_{j\in C_i} G(D(\mathbf{X};\mathbf{R}^j),\sigma)}{\sum_j G(D(\mathbf{X};\mathbf{R}^j),\sigma)}; \quad (5)$$

$$G(D(\mathbf{X};\mathbf{R}^j),\sigma) = \max\left(0, 1-D(\mathbf{X},\mathbf{R}^j)/\sigma\right) \quad (6)$$

Here $W(D) = G(D(\mathbf{X},\mathbf{R}^j);\sigma)$ is the weight associated with the distance D. Reference vectors outside of the σ radius have no influence on the classification probability while their influence inside this radius depends linearly on the distance D. Combining this weighting with the restriction on the number of neighbors leads to the weight $W(D) = \max(0, 1-D/\alpha r_k)$, where r_k is the distance to the $k-$th neighbor and α is an adaptive parameter optimized on the test set.

More sophisticated versions of this algorithm include optimization of the shape of $G(D;\sigma)$ weighting functions using additional parameters. One example is a combination of two sigmoidal functions $\sigma(||\mathbf{X}-\mathbf{R}^j||-b)-\sigma(||\mathbf{X}-\mathbf{R}^j||-b')$, providing larger area in which the weighting factor is essentially constant. Another example is the hyperbolic weighting scheme:

$$p(C|X;M) = \frac{\sum_j \delta(C(X),C)/\left(D\left(\mathbf{X},\mathbf{R}^j\right)+\varepsilon\right)}{\sum_j 1/\left(D\left(\mathbf{X},\mathbf{R}^j\right)+\varepsilon\right)} \quad (7)$$

where ε is a small positive number.

In the Gaussian classifier [11] or in the original RBF network only one parameter σ was optimized [12]. Optimization of the positions of the reference centers $\mathbf{R}^j$ leads to the LVQ method [13] in which the training set is used to define the initial prototypes and the minimal distance rule to assign the classes. The Restricted Coulomb Energy (RCE) classifier [14] uses a hard-sphere weighting functions. The Feature Space Mapping model (FSM) is based on separable, rather than radial weighting functions [15]. All these models are special cases of general SBM framework.

An important problem with localized description of the data by RBF and similar methods concerns the representation of oblique probability distributions of the classes. On very recently a method to create obliqe probability distributions in N-dimensional space using only N parameters has been described [5]. Oblique decision borders in SBM are obtained by rotation of the local coordinate system in which distances are computed. It is sufficient to use a rotation matrix with scaling factors $R_{ii} = s_i$ on the diagonal and rotation parameters $R_{ii+1} = \beta_i$ as the only off-diagonal element.

2.2 D-MLP model

Threshold neurons compute distances in a natural way. If the input signals $\mathbf{X}$ and the weights $\mathbf{W}$ are $(\pm 1 \ldots \pm 1)$ vectors, neuron with N inputs and the threshold θ realizes the following function:

$$\Theta(\sum_i^N W_i X_i - \theta) = \begin{cases} 0 & \text{if } ||\mathbf{W}-\mathbf{X}|| > (N-\theta)/2 \\ 1 & \text{if } ||\mathbf{W}-\mathbf{X}|| \leq (N-\theta)/2 \end{cases} \tag{8}$$

where $||\cdot||$ norm is defined by the Hamming distance (counts the number of mismatches for binary strings). One can interpret the weights of neurons in the first hidden layer as addresses of the reference vectors in the input space and the activity of threshold neuron as activation by inputs falling into a hard sphere of radius $(N-\theta)/2$ centered at $\mathbf{W}$. Changing binary into real values and threshold into sigmoidal neurons for inputs normalized to $||\mathbf{X}|| = ||\mathbf{W}|| = \mathbf{1}$ leads to a soft activation of neuron by input vector close to $\mathbf{W}$ on a unit sphere. The Hamming neural network [16] is actually a neural realization of the nearest neighbor method for a single neighbor and binary vectors.

In general treating $\mathbf{W}$ and $\mathbf{X}$ as vectors and activation as a scalar product $\mathbf{W}\cdot\mathbf{X}$, the activation of a neuron is written as:

$$\mathbf{W}\cdot\mathbf{X} = \frac{1}{2}\left(||\mathbf{W}||^2 + ||\mathbf{X}||^2 - ||\mathbf{W}-\mathbf{X}||^2\right) \tag{9}$$

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

$$\sigma(\mathbf{W}\cdot\mathbf{X}+\theta) = \sigma(d_0 - D(\mathbf{W},\mathbf{X})) \tag{10}$$

where $D(\mathbf{W},\mathbf{X})$ is proportional to the square of Euclidean distance between $\mathbf{W}$ and $\mathbf{X}$ and $d_0 = \frac{1}{2} + \frac{1}{2}||\mathbf{W}||^2 + \theta$. Normalization $||\mathbf{X}|| = 1$ is necessary to avoid the dependence of d_0 on $\mathbf{X}$. Sigmoidal function evaluates the influence of the reference vectors $\mathbf{W}$ on the classification probability $p(C_i|\mathbf{X};\{\mathbf{W},\theta\})$. It plays a role of the weight function $G(D) = \sigma(d_0 - D(\mathbf{W},\mathbf{X}))$, monotonically decreasing, with flat plateau for small distances, reaching the value of 0.5 for $D(\mathbf{W},\mathbf{X}) = d_0$ and approaching zero for larger distances. For normalized $\mathbf{X}$ but arbitrary $\mathbf{W}$ the range of the sigmoid argument lies in the $[\theta - |\mathbf{W}|, \theta + |\mathbf{W}|]$ interval. A unipolar sigmoid has a maximum curvature around ± 2.4, therefore small thresholds and weights mean that the network operates in an almost linear regime. Regularization methods add penalty terms to the error function forcing the weights and thresholds to become small and thus smoothing the network approximation.

From the SBM point of view in MLP networks sigmoidal functions are used to estimate the influence of weight vectors according to the distance between weight and training vectors. By changing the distance function in equation (10) from the square of the Euclidean distance to some other distance measures new types of neural networks, called further D-MLP networks, are defined. Another possibility is to write the weighted product in a form:

$$\sigma(\mathbf{W}\cdot\mathbf{X}) \;=\; \sigma\left(\frac{1}{4}(||\mathbf{W}+\mathbf{X}||^2-||\mathbf{W}-\mathbf{X}||^2)\right) \tag{11}$$

The D-MLP networks simply replace the square of the Euclidean distance in the equation above or in Eq. (10) by Minkovsky's or other type of norms. The network with the nodes computing $\sigma(d_0 - D(\mathbf{W},\mathbf{X}))$ is trained like the standard MLP, using the backpropagation method [1]. Backpropagation procedure requires derivatives of the distance functions, but for Minkovsky and other popular distance functions they are easily derived.

In Eq. (10) the parameter d_0 should be treated as an adaptive parameter only if $\mathbf{X}$ is normalized. This may always be done without loss of information if one or more additional components are added to the vector, extending the feature space by at least one dimension. In particular taking $x_r = \sqrt{R^2 - ||\mathbf{X}||^2}$, where $R \geq \max_X ||X||$, amounts to a projection of the data on a unit hemisphere with radius R (more sophisticated projection is described in [18]). If non-Euclidean norm is used the sphere changes its shape (see sect. 5).

2.3 *Other examples of neural methods derived from SBM framework*

The non-Euclidean D-MLP networks described above are only one of many methods that may be derived from SBM framework. Adapting the similarity function to minimize in-class distance variance and maximize between-class variance, a non-linear version of Fishers discrimination analysis is obtained. Combination of sigmoidal functions offers a good parametrization and network realization here:

$$d(\mathbf{A}_i,\mathbf{B}_i) = \sum_j \alpha_{ij}\sigma\left(\beta_{ij}|\mathbf{A}_i - \mathbf{B}_i| - \gamma_{ij}\right) \tag{12}$$

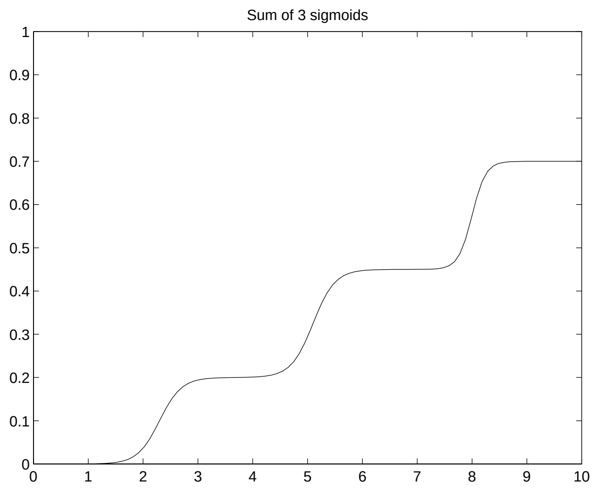

Figure 1: Sum of 3 sigmoidal functions provides a useful distance function allowing to minimize in-class and maximizes between class distances, defining non-linear version of Fisher discrimination.

A network of nodes computing such distances may be used for classification or prediction as any other neural network. It also may be used for extraction of logical rules from data, either fuzzy rules or – in the limit of high slopes – crisp logical rules.

Another interesting possibility is to use a neural network to learn the most appropriate similarity and weighting function. The advantage of such an approach is that instead of having a global coordinate system with a single distance function local coordinate systems smoothly changing in different points of the feature space are defined, providing optimal distance and weighting in different regions of the input space. Combining the *k*-NN approach with neural distance function is certainly worth trying.

Discrimination and cluster-based methods are deeply connected. A single discrimination hyperplane (used in linear discrimination method or provided by MLP neuron) may be replaced by a fixed reference vector (for example cluster center) and an adaptive reference vector. In the Statlog project comparing 20 classification methods the simplest nearest neighbor method appeared as the top one in about one third of all cases [17]. The network realization described below is a generalization of the *k*-NN method and should improve the results on the remaining problems.

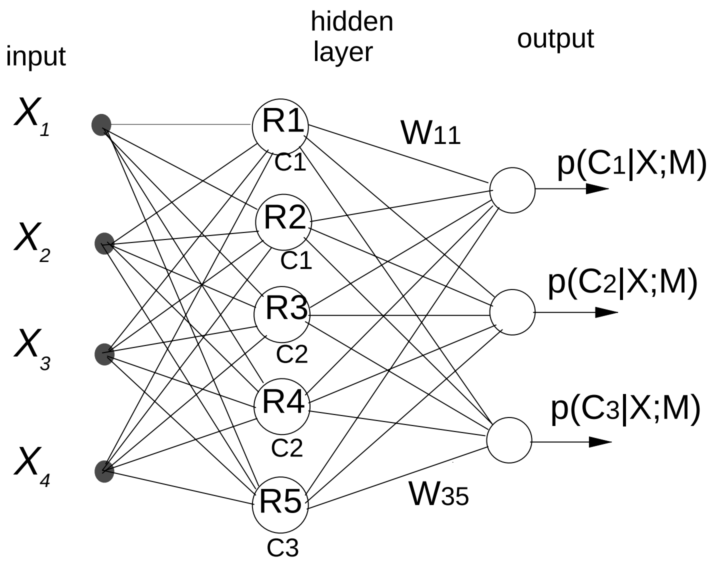

Figure 2: Network generalization of the *k*-NN method. The hidden nodes compute distances to reference vectors and return *k* values of class labels associated with the nodes, while the output nodes compute probabilities.

The network has hidden nodes computing distances $D(\mathbf{X}-\mathbf{R})$, where $\mathbf{R}$ are reference (training) vectors. k nodes with the smallest distances output their class label $h_j(\mathbf{X};\mathbf{R}) = C_i$ and the remaining nodes output $h_j(\mathbf{X};\mathbf{R}) = 0$. The classes are numbered from $C_i = 1\ldots N_C$. The output layer computes probabilities using the formula:

$$P(C_i|\mathbf{X};M) = \sum_j \mathbf{W}_{ij} \cdot h_j(\mathbf{X}) \tag{13}$$

$$p(C_i|\mathbf{X};M) = \frac{P(C_i|\mathbf{X};M)}{\sum_j P(C_j|\mathbf{X};M)} \tag{14}$$

The weight W_{ij} between the hidden node R_j belonging to class C_j and the output node computing probabilities for class C_i is initially equal to $W_{ij} = (1 - \mathcal{R}(C_i,C_j))/C_j$, where the elements of the risk matrix $0 \leq \mathcal{R}(C_i,C_j) \leq 1$ in the simplest k-NN are replaced by $delta_{ij}$. Thus each vector that belongs to the k nearest ones or that falls into the r radius of $\mathbf{X}$ and is of the C_j class, contributes to the probability of the C_i class a value $1 - \mathcal{R}(C_i,C_j)$. The structure of the network is shown in Fig. 2. For the cost function that should be optimized one may take:

$$E(\mathbf{W};M) = \sum_{\mathbf{X},i} \left(p(C_i|\mathbf{X};M) - \delta(C_i,C(\mathbf{X}))\right)^2 \tag{15}$$

where the model M includes k, weights and distance-related as parameters. If the number of classification errors should be minimized binary 0, 1 output probabilities are taken, provided for example by the winner-takes-all neural procedure. Binary probabilities should be used with global minimization or search-based methods, since gradient-based methods cannot be used in this case. The output weights, initialized to $W_{ij} = (1 - \mathcal{R}(C_i,C_j))/C_j$, may be treated as adaptive parameters. Introduction of soft weighting $G(D(\cdot))$ allows to use gradient optimization methods. For many datasets (especially for images [17]) this simple network should outperform MLPs and other classification models, since the results should be at least as good as the k-NN results.

A single neuron provides discrimination hyperplane which may be replaced by one reference vector. Position of this reference vector should be adapted to the data. Using different Minkovsky distance functions changes the decision borders. Using one prototype $\mathbf{R}_i$ per class (i.e. one hidden node) class membership is decided by discriminant function:

$$z(\mathbf{X}) = W_1 D(\mathbf{X},\mathbf{R}_1) - W_2 D(\mathbf{X},\mathbf{R}_2) - \theta \tag{16}$$

where θ is a threshold. The 3 adaptive parameters, W_1, W_2, θ and the positions of two prototype vectors provide very flexible decision borders in the two class problem. If more reference vectors are required the output node computing discriminating function sums over prototypes for each class:

$$z(\mathbf{X}) = \sum_{i \in C_1} W_i D(\mathbf{X},\mathbf{R}_i) - \sum_{i \in C_2} W_i D(\mathbf{X},\mathbf{R}_i) - \theta \tag{17}$$

Scaling of the whole sum, instead of influences of individual reference vectors, is a simple way to reduce the number of adaptive parameters used by the system. One option that we are investigating is to use simple gradient optimization for weights and thresholds, and search based techniques for non-linear scaling parameters.

Similarity of such neural realization of the nearest neighbor method to RBF model with radial coordinate functions should be noted. If the number of neighbors is not restricted the two methods are identical.

SBM point of view on neural networks not only allows to define many new methods, but leads also to novel applications such as pattern completion or associative memory recall. A natural cluster-based initialization described below determines all parameters of D-MLP networks.

3 Initialization of the network

The D-MLP network uses normalized vectors, adding one extra dimension if necessary, projecting the data on a hemisphere. The network should be initialized taking the centers of

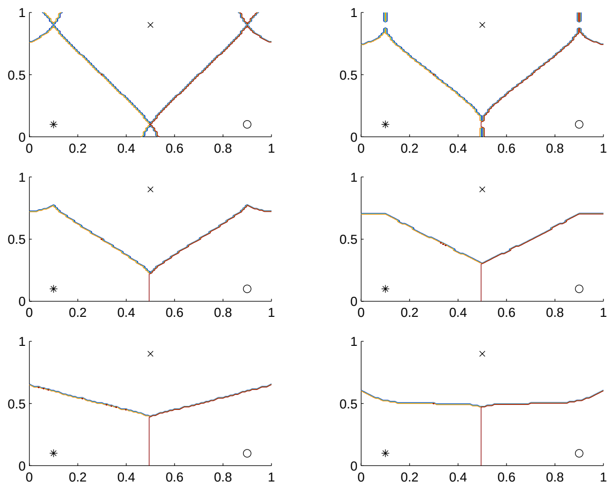

Figure 3: Decision borders for various exponents of Minkovsky distance function in the nearest neighbor method for α=0.1, 0.3, 0.7, 1, 2, 8. All weights are identical.

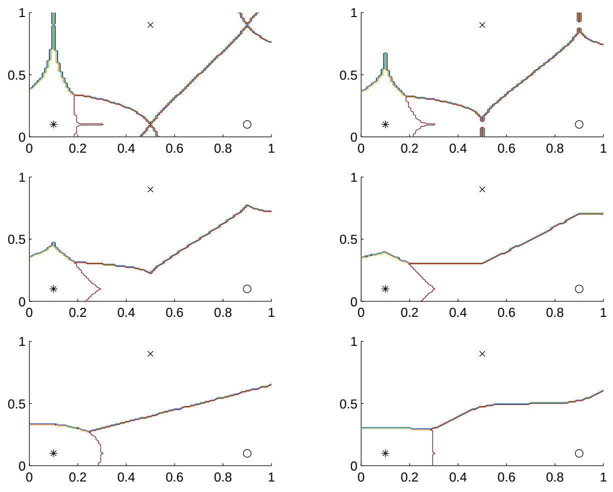

Figure 4: Decision borders for various exponents of Minkovsky distance function in the nearest neighbor method for α=0.1, 0.3, 0.7, 1, 2, 8. Weight of the first prototype is 3 times larger than other weights.

clusters in the extended $N+1$ dimensional space as weights $\mathbf{W}$, and taking $d_0 = D(\mathbf{W}, \mathbf{X}^b)$, where $\mathbf{X}^b$ is a vector at the border of a given cluster. To define clusters we have tried [18] dendrograms and decision trees, but other clusterization methods may also be used for initialization [11]. Using weighted activation the contribution of a center of an input data cluster $\mathbf{C}$ laying on the unit sphere is $\mathbf{W} \cdot \mathbf{C}$. The largest activation is obtained when the weights $\mathbf{W}$ point in the same direction as the center $\mathbf{C}$. The sigmoidal function $\sigma(\mathbf{C} \cdot \mathbf{X} - \theta) = (1 + \exp((-\mathbf{C} \cdot \mathbf{X} + \theta)/T))^{-1}$, where T determines the slope, has the largest gradient in the direction of $\mathbf{W} = \mathbf{C}$. The value $\sigma(0) = 0.5$ is obtained at a θ distance from the origin of the coordinate system. Since the $\mathbf{C}$ vector is normalized $\theta = 1$ places the contours for 0.5 value tangentially to the unit hypersphere. Contours for lower values $\sigma(\mathbf{C} \cdot \mathbf{X} - \theta) < 0.5$ cut segments of the hypersphere in which the value of $\sigma(\mathbf{C} \cdot \mathbf{X} - \theta)$ is constant.

A parameter which is rarely changed in MLPs is the slope of sigmoidal functions. It defines the area which has an influence on performance of each node. If the slope is too high the area in which the sigmoidal function is not approximately constant is small and only a few training vectors have a chance to influence the gradient-based learning procedures. If it is too low then all functions strongly overlap and there is no possibility to create sharp decision borders. Normalization of the weights $\mathbf{W}$ is equivalent to a local change of the slope:

$$\begin{aligned} (\mathbf{W} \cdot \mathbf{X} + \theta)/T &= \left(\frac{\mathbf{W}}{||\mathbf{W}||} \cdot \mathbf{X} + \frac{\theta}{||\mathbf{W}||}\right)||\mathbf{W}||/T \\ = (\mathbf{W}' \cdot \mathbf{X} + \theta')/T' &= (d_0' - D(\mathbf{W}', \mathbf{X}))/T' \end{aligned} \qquad (18)$$

Thus without loss of generality both $\mathbf{X}$ and $\mathbf{W}'$ may be normalized. No special learning for the slopes is required. A useful variability range of the sigmoid is between its maximum curvature points, which for $T = 1$ are between $\Delta(T) = \pm 2.4$. If the variability range is assumed to be 1/10 of the size of the cluster, i.e. $\Delta(T) = \pm d_0/10$ then setting $T \approx d_0/24$ will be appropriate. After such initialization training of the network is usually quite short.

In practice one may take a dendrogram, starting from the top few largest clusters, initializing the network with the number of neurons equal to the number of these clusters, and training the network. If the complexity of the network is too low results on the training set will be poor. More clusters should be taken into account – in case of dendrograms one should break those clusters that are inhomogenous in the first place. More complex network is initialized and trained, until the results on the training set will be satisfactory.

In the XOR case the input vectors for class = T are $(0,1), (1,0)$ and for the class = F are $(0,0), (1,1)$. The mean for each feature is 0.5 and after shifting and renormalizing the vectors are $\mathbf{C}_1 = (-1, +1)/\sqrt{2}$, $\mathbf{C}_2 = (+1, -1)/\sqrt{2}$ for class T and $(-1, -1)/\sqrt{2}$, $(+1, +1)/\sqrt{2}$ for class F. Selecting one of the classes for output, for example class T, initial weights for the first neuron are given by $\mathbf{C}_1$ and for the second neuron by $\mathbf{C}_2$, while the hidden to output layer weights are all $+1$. This is the correct and the simplest solution for the XOR problem found without any optimization of the network! For more complex examples of this type of initialization see [18].

Since the architecture of the MLP network in the extended space is completely determined by the initialization procedure (the clusterization method used determines all parameters) and the training is short due to a good starting point many distance functions may be tried on a given problem.

4 Pattern completion, associative memory and missing values.

Methods belonging to the SBM framework, such as the nearest neighbor method, may be used as associative memories in a natural way. Any part of the input vector $X = (X_d, X_u)$ may be used to find nearest neighbors in the subspace of defined input values X_d. The undefined part X_u is predicted interpolating the values of nearest neighbors for the dominating class. Optimization of parameters for classification in the X_d subspace only should improve results but frequently the same k-NNmodel works well in subspaces.

Pattern completion may be implemented in several ways. In many cases vectors with missing values are removed from the training set or some averaged or most frequent values are inserted. In this way useful information is thrown out or inappropriate information is introduced. For example, the echocardiogram data from UCI repository [20], contains 132 vectors, 12 attributes of which only 1-9 are useful, the second being the class. 15 values of the attribute 6 are missing, 11 values for attribute 7 etc. If the attributes with missing values are ignored 10-fold stratified crossvalidation tests gives 87.8% accuracy using on average 24 neurons of the FSM network [15] (FSM is based on constructive algorithm, therefore different number of neurons may be created in different crossvalidations), while inserting averages over all classes decreased the accuracy to 85.5% (with 20 neurons), and inserting a new value that does not appear in the data, such as -100, decreased accuracy to 81.5% (using 22 neurons). The same behavior has been observed for Hepatitis dataset taken from the same source. the data contains 155 vectors, 18 attributes, 13 of them are binary, other have integer values. The last attribute has 67 missing values, attribute 16 has 29 missing values etc. Using 10-fold crossvalidation tests ignoring missing values gives 79.9% accuracy using on average 19 neurons, inserting averages over all classes 81.0% (with 12 neurons) and inserting -100 gives lowest accuracy 79.1% (with 16 neurons).

Suppose that 2-dimensional data vectors are clustered around (1.0,1.0) and (2.0,2.0), with the first cluster containing twice as many vectors as the second. Suppose now that the second feature is missing in the training vector X with $x_1 = 1.9$. If X neighbors in the x_1 subspace, around the given $x_1 = 1.9$ values are found, interpolating the missing x_2 value will give approximately correct answer (around 2.0) while using the most frequent values or averaged values will give incorrect guess (around 1.0). In many applications hierarchical approach to collection of data is taken: initial tests allow to make a hypothesis, followed by specific tests that confirm it or not. The challenge is to discover such hierarchical classification. In statistics analysis of independent surveys in which some questions are not answered by some respondents and some questions are not asked in some surveys is known as the "multiple imputation" problem (see [21]), but assumptions about normal distributions used in this theory may not be valid. Another approach is described below.

In the first step missing features in the training vectors should be completed. Information contained in training vectors with missing features is than used to improve the classification model. Probability of unknown values X_u is calculated by maximization of:

$$p(X_u|X_d;M) = \max_{u',i} p(C_i|(X_{u'}, X_d);M) \tag{19}$$

i.e. searching for the maximum of the probability given by the model M in the subspace of undefined features, with fixed point in the X_d subspace. If a single missing feature is sought one dimensional maximization or a search procedure in the range of admissible values for X_u is done. Initial model M is prepared using either training vectors that have all features defined, or – if most vectors contain missing values – a largest subset of training vectors is found with the largest number of the same input features defined. For example, if only a few vectors with all values are given but a large number contains just a single missing value X_u, the initial classification model should be based on reduced number of features. The model is then retrained using vectors containing the extra feature X_u and the missing values of this feature imputed to the remaining vectors. At each step one may check if it is worth to include the new feature and to perform pattern completion. If the results in crossvalidation tests get worse the feature should be dropped.

For strongly interacting features the problem of initial feature selection and the order of imputing/adding features suffers from combinatorial explosion. There is no guarantee that the optimal model will be found. In practice network computations make the whole search procedure rather simple, since after a reasonable initial model is created maximization in Eq. (19) does not involve costly multidimensional searches, but can be performed analytically or by evaluating the excitation level of network nodes. Moreover, since networks offer analytical representation of computed probabilities integration using statistical sampling techniques is easily performed. Using FSM network and the method based on Eq. (19) for the two datasets mentioned above we have obtained for the echocardiogram 90.2% using only 18 neurons, and for the hepatitis 83.4% accuracy using only 10 neurons, significantly better result than by other methods.

5 Normalization of input vectors in non-Euclidean spaces

The parameter d_0 should be treated as an adaptive parameter only if $\mathbf{X}$ is normalized. This may always be done without loss of information if one or more additional components are added to the vector, extending the feature space by at least one dimension. Taking $X_r = \sqrt{R^2 - ||\mathbf{X}||^2}$, where $R \geq \max_X ||\mathbf{X}||$, amounts to a projection of the data on a unit hemisphere with radius R. In general vectors $(\mathbf{X}, X_r)$ may be normalized $||(\mathbf{X}, X_r)||_D = 1$ using the metric defined by the distance function $D(\mathbf{X}, \mathbf{R})$.

The distance function may be heterogeneous, using Minkovsky's metric for numerical features and probabilistic metric functions for symbolic features. Minkovsky's distance with the scaling factors is:

$$D(\mathbf{A}, \mathbf{B}; s)^\alpha = \sum_i^N s_i d(\mathbf{A}_i, \mathbf{B}_i)^\alpha \tag{20}$$

The $d(\cdot)$ function is used to estimate similarity at the feature level and in the simplest case is equal to $|A_i - B_i|$. For large α this metric changes the sphere into a soft cuboid, for $\alpha = 1$ it becomes a pyramid and for $\alpha < 1$ it has a hypocycloidal shape. Instead of deriving the backpropagation equations for the transfer functions with non-Euclidean distances one may achieve similar result using a standard MLP network with x_r determined by the normalization condition using the desired metric.

In memory-based reasoning the Modified Value Difference Metric (MVDM) has gained some popularity [19]. The distance between two N-dimensional vectors A, B with discrete (nominal, symbolic) elements, in a K class problem, is computed using conditional probabilities:

$$D_\alpha(A, B) = \sum_j^N \sum_i^K \left| p(C_i|A_j) - p(C_i|B_j) \right|^\alpha \tag{21}$$

where $p(C_i|A_j)$ is estimated by calculating the number $N_i(A_j)$ of times the value A_j of the feature j occurred in vectors belonging to class C_i, and dividing it by the number of times A_j occurred for any class. A "value difference" for each feature j is defined as $d_V^\alpha(A_j, B_j) =$

$\sum_i^K |(p(C_i|A_j) - p(C_i|B_j))|^\alpha$. It allows to compute $D_V(\mathbf{A},\mathbf{B})$ 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 continuos values requires a set of probability density functions $p_{ij}(x)$, with $i = 1..K, j = 1..N$.

Using VDM type of metrics leads to problems with calculation of gradients, therefore another method is advocated here. Replacing symbolic features by vectors of $p(C_i|A_j)$ probabilities (with dimension equal to the number of classes times the number of different symbolic values the feature takes) allows to reproduce MVDM distances using numerical values of vector components. Many other types of metric functions exist [19] and their performance should be empirically verified. Several alternative extensions of the input space may be considered, for example adding one or more features $X_r = D(\mathbf{X},\mathbf{R})$ equal to the distance of a given vector $\mathbf{X}$ to some fixed vector $\mathbf{R}$ a parabolic projection is made.

It may be of some advantage to increase the separation of the clusters projected on the hypersphere. It is impossible to make such a projection on the whole hypersphere without violating topological constraints. In the one-dimensional case with $X \in [-1,+1]$ the (X, X_r) vector should not make a full circle when X is changed from -1 to $+1$ because the two extreme vectors $X = \pm 1$ will then be identical. An optimal separation for 3 vectors with the length $||X||, ||X|| + \Delta, ||X|| + 2\Delta$ is to place them in corners of equilateral triangle, for example at angles $0, \pm 120°$. One can search for the best input preprocessing treating it as a rigorous optimization problem, or just use polar coordinates to shift some upper hemisphere vectors to the part of the lower hemisphere. Much simpler approach is to rescale all vectors to get their Euclidean norms ≤ 1, use the norm $||X||$ mapping it to points on a circle: $\left(\sin\frac{\pi}{3}(4-5||X||), \cos\frac{\pi}{3}(4-5||X||)\right)$. These points for $0 \leq ||X|| \leq 1$ are within the angle $-\pi/3$ and $4\pi/3$. The first factor, $\sin\frac{\pi}{3}(4-5||X||)$ is used to rescale components of the vector $\mathbf{X}$, while the second factor is taken as an extra X_r component. Extended vectors $||(\mathbf{X}^j, X_r^j)||_D$ are renormalized using the metric function $D(\cdot)$, placing them on a unit sphere defined by this metric.

6 Pedagogical illustration

The influence of non-Euclidean distance functions on the decision borders is illustrated here on the classical Iris flowers dataset, containing 50 cases in each of the 3 classes. The flowers are described by 4 measurements (petal and sepal width and length). Two classes, Iris virginica and Iris versicolor, overlap, and therefore a perfect partition of the input space into separate classes is not possible. An optimal solution (from the point of view of generalization) contains 3 errors and is obtained using only two of the four input features (x_3 and x_4), therefore it is easy to display and only those two features have been left in simulations described below.

A standard MLP solution is obtained with 2 input, 4 hidden and 3 output neurons, with a total of 27 adaptive parameters. One discriminating plane per class for the smallest and the largest flowers (setosa and virginica) is needed and two planes are needed to separate the vectors of the versicolor class. To increase accuracy and speed up learning, in the final phase of learning only the vectors near the class borders were presented to the network. The selection algorithm loops over all vectors and for a given vector $\mathbf{X}$ finds k (for example $k = 10$) nearest vectors belonging to a different class than $\mathbf{X}$. These vectors are written to a new training file providing a description of the border region. This method of training leads to sharper and more accurate decision borders, as seen in the first drawing of Fig. 6.

An additional input feature has been added and the 3-dimensional vectors normalized

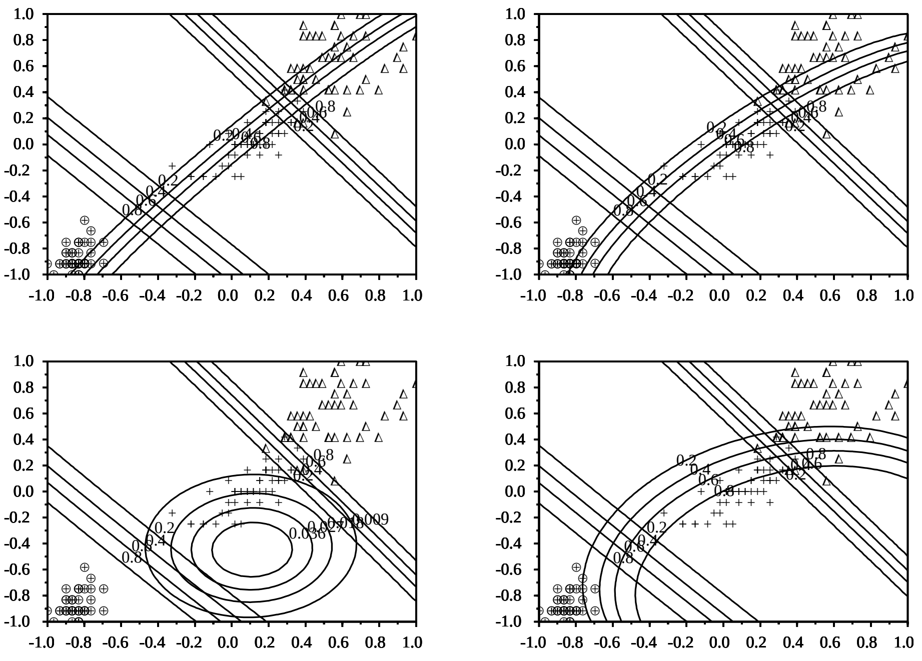

Figure 5: Shapes of decision borders in the Iris case for the network without the hidden layer (3 output neurons, 3 inputs), with growing W_3 weight.

using various Minkovsky distance measures. The network has been initialized taking the normalized weights that are equal to the centers of the three clusters. In the extended feature space the same accuracy is achieved using only 3 input and 3 output neurons without the hidden layer, for a total of 12 adaptive parameters. Using such network architecture with squared Euclidean metric and the weight $W_3 = 0$ for the third X_3 component, only two classes are separated. The transition process from the planar to circular decision borders is shown in Fig. 6 (clockwise, from top left). In the learning process W_3 grows, curving the borders for vectors near the center of the drawing.

The data has been standardized and rescaled to fit it inside a square with ± 1 corners. An additional feature has been added and the 3-dimensional vectors normalized using various Minkovsky distance measures. The network has been initialized taking the normalized weights that are equal to the centers of the three clusters. In the extended feature space only 3 neurons are necessary. In Figure 6 dramatic changes in the shapes of decision borders for Minkovsky metric are observed. Using squared Euclidean metric in $\sigma(d_0 - D(\mathbf{X}, \mathbf{R}))$ transfer functions 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 a hypocycloidal shapes are created. Since smooth transition between these cases is made α should be treated as an adaptive parameter. For the Iris data the optimal solution (3 errors) has been recovered for all values of $\alpha \geq 0.8$. Smooth transition between these cases is made by changing α and retraining the network. For other datasets we have found significant improvements of accuracy for optimized α.

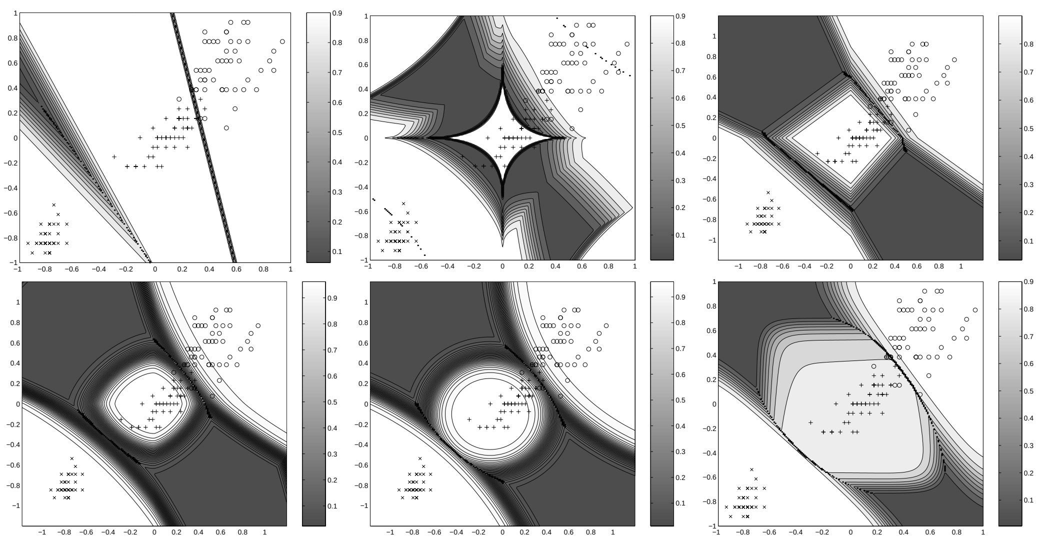
Figure 6: Shapes of decision borders in the Iris case for standard MLP (3 neurons) and an MLP network in the extended space with vectors renormalized using Minkovsky metric, $\alpha = 0.5, 1.0, 1.5, 2.0$ and 7.0.

7 Discussion

Building robust neural networks, as fast and as easy to use as decision trees is still a challenge. New neural models described here, such as the D-MLP networks with non-Euclidean distance functions, are one of many realizations of similarity based methods. Non-Euclidean transformation of input vectors leads to very flexible shapes of neural network decision borders without any change in the standard computer programs. The training times are short since a good initialization procedure based on clusterization techniques determines weights, thresholds and the slopes of all neurons. The number of neurons in the network defined in extended space may also decrease, as has been observed in the Iris example. A new method to treat symbolic values and a new training procedure using only the vectors close to the decision borders have been used here. Since the training is fast many different metric functions may be tried before selecting (using crossvalidation tests) the best model. Networks with activation given by Eq.(10) or (11) have not yet been implemented but such models seem to be quite promising.

The change of the shapes of decision borders has been accomplished before by adding new type of units to neural networks. For example, Ridella *et al.* [22] used circular units in their Circular Backpropagation Networks. Different type of circular units have been used by Kirby and Miranda [23] – in their implementation two sigmoidal units are coupled together and their output is restricted to lie on a unit circle. Dorffner [24] proposed conic section transfer functions as a unified framework for MLP and RBF networks. Straight lines and ellipses are special cases of conic sections. The method presented here may be treated as a generalization of the circular or conical unit method. It is not restricted to MLP neural networks, but can be used with any neural network and any classifier.

Neural models derived from the SBM framework solve not only classification, but also patter completion and associative memory problems. An additional advantage of the approach outlined here is the understanding of what these networks have really learned in terms of the prototypes (weights) and the weighted distances from these prototypes. A challenge for the neural networks is combinatorial productivity and extraction of knowledge from complex datasets. First steps in this direction have already been done [15, 4].

Acknowledgments: Support of W.D. and R.A. by the Polish Committee for Scientific Research, grant 8T11F 014 14, is gratefully acknowledged.

References

[1] C. Bishop, *Neural networks for pattern recognition*. Clarendon Press, Oxford, 1995.

[2] C. Cortes and V. Vapnik, Support Vector Networks. *Machine Learning* **20** (1995) 273-297

[3] W. Schiffman, M. Joost, R. Werner, Proc. of ESANN'93, Brussels 1993, pp. 97-104

[4] Duch W, Adamczak R, Grąbczewski K, Methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks* (submitted, June 1999)¡/LI¿

[5] W. Duch, N. Jankowski, New neural transfer functions. *Applied Mathematics and Computer Science* **7** (1997) 639-658

[6] D.M. Skapura, *Building neural networks*, Addison-Wesley, 1996.

[7] X. Yao, A Review of evolutionary neural networks, *International Journal of Intelligent Systems* **8** (1993) 539-567

[8] Duch W, Adamczak R, Grąbczewski K, Extraction of logical rules from backpropagation networks. *Neural Processing Letters* **7** (1998) 1-9

[9] W. Duch, Neural minimal distance methods. Proc. 3-rd Conf. on Neural Networks and Their Applications, Kule, Poland, Oct. 14-18, 1997, pp. 183-188

[10] L. Breiman, Bagging predictors, *Machine Learning* **24** (1996) 123-140

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

[12] P.D. Wasserman, *Advanced methods in neural networks*. Van Nostrand Reinhold, 1993.

[13] T. Kohonen, *Self-organizing maps*. Berlin, Springer-Verlag, 1995.

[14] D.L. Reilly, L.N. Cooper, C. Elbaum, A neural model for category learning, *Biological Cybernetics* **45** (1982) 35–41

[15] W. Duch, G.H.F. Diercksen, Feature Space Mapping as a universal adaptive system. *Comp. Phys. Communic* **87** (1995) 341-371

[16] R.P. Lippmann, An introduction to computing with neural nets. *IEEE Magazine on Acoustics, Signal and Speech Processing* **4** (1987) 4–22

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

[18] W. Duch, R. Adamczak, N. Jankowski, Initialization and optimization of multilayer perceptrons, 3rd Conf. on Neural Networks and Their Applications, Kule, Poland, October 1997, pp. 105-110

[19] D.R. Wilson, T.R. Martinez, *Improved heterogenous distance functions*. J. Artificial Intelligence Research **6** (1997) 1-34

[20] C.J. Mertz, P.M. Murphy, UCI repository of machine learning databases, http://www.ics.uci.edu/pub/machine-learning-data-bases.

[21] D.B. Rubin, Multiple imputation after 18+ years. *J. of the American Statistical Association* **91** (1996) 473-489

[22] S. Ridella, S. Rovetta, R. Zunino, Circular Backpropagation Networks for Classification, *IEEE Trans. Neural Networks* **8** (1997) 84–97

[23] M.J. Kirby, R. Miranda, Circular Nodes in Neural Networks, *Neural Computations* **8** (1996) 390-402

[24] G. Dorffner, A Unified Framework for of MLPs and RBFNs: Introducing Conic Section Function Networks, *Cybernetics & Systems* **25** (1994) 511-554