Learning highly non-separable Boolean functions using Constructive Feedforward Neural Network

Marek Grochowski and Włodzisław Duch

Department of Informatics, Nicolaus Copernicus University,
Grudziądzka 5, Toruń, Poland,
grochu@is.umk.pl; Google: Duch

Abstract. Learning problems with inherent non-separable Boolean logic is still a challenge that has not been addressed by neural or kernel classifiers. The k-separability concept introduced recently allows for characterization of complexity of non-separable learning problems. A simple constructive feedforward network that uses a modified form of the error function and a window-like functions to localize outputs after projections on a line has been tested on such problems with quite good results. The computational cost of training is low because most nodes and connections are fixed and only weights of one node are modified at each training step. Several examples of learning Boolean functions and results of classification tests on real-world multiclass datasets are presented.

1 Introduction

Many algorithms and neural network architectures for learning parity problem and other difficult Boolean functions were presented in [1–5]. These methods are suitable only for the parity problem and will not work for other complex problems. Biological neural networks are able to solve quite complex learning problems inherent in optimization of behavior or understanding linguistic patterns. Finding general algorithm capable of solving a larger set of problems of similar complexity as the parity problem is still a challenge. Sequential constructive methods using computational geometry algorithms are the most promising in this respect. Several interesting algorithms of this type, including the irregular partitioning, Carve, target switch, oil spot, and sequential window learning algorithm, have been reviewed and compared in [6]. Most of them work only for binary problems and therefore pre-processing using Gray coding is used.

As shown recently [7] complexity of classification problems may be characterized by the concept of k-separability: the dataset $\boldsymbol{x}$ of points belonging to two classes is called k-separable if a direction $\boldsymbol{w}$ exist such that k is the minimum number of intervals containing data from a single class after projection on a line $y_i = \boldsymbol{x}_i \cdot \boldsymbol{w}_i$. For example, linearly separable two-class data points form just two intervals (open-ended on left and right side), while the data that cannot be linearly separated may require 3 or more intervals containing vectors from a single class only. The n-bit parity problems require at least $k = n + 1$ intervals. This concept allows for precise characterization of the complexity of

learning. Although usually the smallest k is of interest sometimes higher k's may be preferred if the margins between projected clusters are larger, or if among k clusters some have very small number of elements. Problems that may be solved by linear projection on at least k-clusters belonging to alternating classes are called k-separability problems [7]. For the parity $k = n + 1$ is sufficient and it is quite likely (although it has not been proven) that all Boolean functions may be learned using linear projection on no more than $n + 1$ intervals.

The difficulty of learning Boolean functions grows quickly with the minimum k required to solve a given problem. Linear (2-separable) problems are quite simple and may be solved with linear SVM or any other variant of LDA model. Kernel transformations may convert some data distributions into linearly separable distributions in higher dimensional space. This strategy does not work for parity-like and other difficult Boolean functions, with virtually no generalization from learned examples. Gaussian-based kernels set each data point as a separate support vector, and in case of parity all points have closest neighbors from the opposite class. Nearest neighbor methods and decision trees have the same problem, linear methods fail completely while multilayer perceptrons, although in principle may be used [1–5], in practice require special network architectures and convergence of the learning procedure is almost impossible to achieve. As already shown in [8] (see also [9, 10]) some problems require at least $O(n^2)$ parameters using networks with localized functions and only $O(n)$ parameters when non-local functions are used. The n-parity problem may be trivially solved using a periodic function with a single parameter [7] while the multilayer perceptron (MLP) networks need $O(n^2)$ parameters and learn it only with great difficulty.

For many complicated problems often a simple linear mapping exists that leaves only trivial non-linearities that may be separated using window-like neurons. For example XOR, the simplest non-linearly separable problem, is solved by a network with one node implementing window-like transfer function:

$$\tilde{M}_i(\boldsymbol{x}; \boldsymbol{w}, a, b) = \begin{cases} 1 \text{ if } \boldsymbol{wx} \in [a, b] \\ 0 \text{ if } \boldsymbol{wx} \notin [a, b] \end{cases} \tag{1}$$

This function is suitable for learning all 3-separable data, and the number of such Boolean functions for 3 or more bits is much greater than of the linearly separable functions [7]. There are many advantages of using window-type functions in neural networks, especially in difficult, highly non-separable classification problems [8].

Problems requiring $k = 3$ are already slightly more difficult for non-local transformations (for example MLPs) and problems with high k quickly become intractable for general classification algorithms. Although some k-separable problems may also be solved using complex models it is rather obvious that simplest linear solutions, or solutions involving smooth minimal-complexity non-linear mappings combined with interval non-linearities, should show better generalization and such solutions should be easier to comprehend.

An initial data transformation by the hidden layer may create an image of the data that projected on some direction $\boldsymbol{w}$ will produce pure clusters of samples

that belong to a single class and can be easily separated from vectors from the opposite class. Unfortunately the standard learning algorithm cannot be used because it is not known a priori to which interval a given vector should be assigned, so labels cannot be used directly as targets for learning. In this paper a simple and fast network with constructive algorithm and hidden nodes with transfer functions of type (1) is proposed to solve this problem. This network is described in the next section, and the Section 3 contains some experimental results on learning the parity problem and other difficult Boolean functions. Some cross-validation test results on benchmark multiclass problems are also presented.

2 The Network Architecture and Training

Different neural network approaches may be used to search for k-separable solutions [7, 11]. For difficult Boolean functions what is needed is a combination of projection and clustering, with localized functions capturing interesting clusters in projections. For 3-separable two-class problems the transformation provided by the network should lead to 3 intervals: $[-\infty, a], [a, b], [b, +\infty]$ to $-1, +1, -1$ values. This may be implemented using a single transfer function of type (1) that separates instances from the interval $[a, b]$ from the rest, grouping vectors from a single class into a cluster. For backpropagation algorithm (and other gradient descent based methods) soft windowed-type functions should be used. Many types of transfer functions can be used for realization of (1) (for taxonomy of a neural transfer functions see [12], [8]). For example, the bicentral function may be used in a product form:

$$M_i(\boldsymbol{x}; \boldsymbol{w}, t, a, \beta) = \sigma(\beta(\boldsymbol{w}\boldsymbol{x} - t - a))(1 - \sigma(\beta(\boldsymbol{w}\boldsymbol{x} - t + a))$$

or taken as a difference of two sigmoidal functions:

$$M_i(\boldsymbol{x}; \boldsymbol{w}, a, b, \beta) = \sigma(\beta(\boldsymbol{w}\boldsymbol{x} - a)) - \sigma(\beta(\boldsymbol{w}\boldsymbol{x} - b))$$

.

These two functions differ for $b < a$, with the second one producing negative output $M(b < a) < 0$. This property may be useful for "unlearning" instances misclassified by previous hidden nodes. Hard-window type function (1) is obtained by setting large value of the slope β of sigmoidal functions at end of the learning process.

$$M_i(\boldsymbol{x}; \Gamma_i) \stackrel{\beta \to \infty}{\longrightarrow} \tilde{M}_i(\boldsymbol{x}; \Gamma_i)$$

or by introduction of an additional threshold parameter:
$\tilde{M}_i(\boldsymbol{x}; \Gamma_i) = \mathrm{sgn}\,(M_i(\boldsymbol{x}; \Gamma_i) - t_i)$. An interesting new window-type function that may be used as a transfer function is:

$$M_i(\boldsymbol{x}; \boldsymbol{w}, a, b, \beta) = \frac{1}{2}\big(1 - \tanh(\beta(\boldsymbol{w}\boldsymbol{x} - a)) \tanh(\beta(\boldsymbol{w}\boldsymbol{x} - b))\big) \ . \qquad (2)$$

It is easy to show that for all possible values of β there is $M(wx = b) = M(wx = a) = 0.5$. Hence the interval boundaries during the learning phase, when the slope β is small, are exactly the same as boundaries taken when the value of β reaches large values – in the bicentral function case interval boundaries for $\beta \to \infty$ may be different than $[t - a, t + a]$.

For more complex problems ($k > 3$) the network with weight sharing based on several such nodes may be used. A standard network with window functions should minimize an error measure:

$$E(\boldsymbol{x}; \Gamma) = E_{\boldsymbol{x}}||y(\boldsymbol{x}; \Gamma) - c(\boldsymbol{x}))|| \tag{3}$$

where $y(\boldsymbol{x}; \Gamma)$ is network output, $c(x) \in \{0, 1\}$ denote class of given vector x and Γ is a set of all parameters that are changed during training (weights, biases, etc.). This expectation is calculated over all vectors, and any norm may be used, including the most popular mean square error or cross entropy measures. The minimum is reached for parameters Γ and if we could find good solution for the whole network the problem would be solved. For complex problems (larger k) this would require global minimization. This form of error does not give any control over purity of the clusters, but adding for each hidden node with window-like transfer function an extra term:

$$E(\boldsymbol{x}; \Gamma; a, b, \lambda) = E_{\boldsymbol{x}}||y(\boldsymbol{x}; \Gamma) - c(\boldsymbol{x}))|| + \lambda E_{y\in[a,b]}||y(\boldsymbol{x}; \Gamma) - c(\boldsymbol{x}))|| \tag{4}$$

will allow to get minimum error for a pure cluster, with λ controlling the tradeoff between the covering and the purity. This approach requires constructive network, with nodes trained one at a time. All experiments described below were done with the following error function:

$$\begin{aligned} E(\boldsymbol{x}; \Gamma, \lambda_1, \lambda_2) = \frac{1}{2}\sum_{\boldsymbol{x}}(y(\boldsymbol{x}; \Gamma) - c(\boldsymbol{x}))^2 + \\ +\lambda_1 \sum_{\boldsymbol{x}}(1 - c(\boldsymbol{x}))y(\boldsymbol{x}; \Gamma) - \lambda_2 \sum_{\boldsymbol{x}} c(\boldsymbol{x})y(\boldsymbol{x}; \Gamma) \end{aligned} \tag{5}$$

First term in (5) is the standard mean square error (MSE) measure, second term with λ_1 (penalty factor) increases the total error for vectors x_i from class $c(x_i) = 0$ that falls into group of vectors from class 1 (it is a penalty for "unclean" clusters), third term with λ_2 (reward factor) decreases the value of total error for every vector x_i from class 1 that was correctly placed inside created clusters (it is a reward for large clusters).

The network used here has one hidden layer and one sigmoidal or linear node in the output layer; it's architecture is shown in Fig. 1.

All connection weights between the output and the hidden layer have fixed strength 1, simply summing the outputs filtered through the window functions. Every hidden node should separate a large group of vectors from class $c = 1$. The

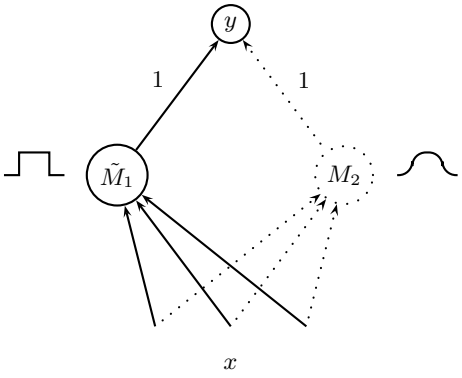

Fig. 1. Example of a network with two hidden neurons. Only parameters of the second node are adapted (dotted line), the first node M_1 has been frozen with large value of β.

weight sharing requirement to obtain linear projections may relaxed, allowing different projection directions for different neurons, although initialization from the same direction may frequently need to linear k-separable solution. Learning starts with an empty hidden layer and in every phase of the training one new unit is added, initialized and trained using the backpropagation algorithm until convergence is reached. Weight sharing is not used to allow for different projections that may lead to partially overlapping clusters. Node weights are initialized with small random values, while biases a and b (for sigmoidal type of neurons) are estimated using $a, b = (\boldsymbol{wx})_{min} \pm \frac{1}{3}|(\boldsymbol{wx})_{max} - (\boldsymbol{wx})_{min}|$. The input vectors correctly handled by the first neuron do not contribute to the error, therefore the weights of this neuron are kept frozen during further learning. Next node is added and learning procedure is repeated on the remaining data. If number of cases correctly classified by a given new node drops below certain minimum the learning procedure stops and this node is removed from the network.

3 Results.

Reward and Penalty. There are 65536 possible 4 dimensional Boolean functions and 192 of them are 6-separable functions [7]. The remaining functions are k-separable with $k < 6$. Hence for learning this functions at least 2 hidden nodes of type (1) are needed. The network have been applied to 192 functions that are 6-separable, using various values of λ_1 and λ_2. Figure 2 shows the effect of reward and penalty on average error, the number of learning cycles and the numbers of nodes needed in the final network to find a perfect solution. While reward λ_2 grows the error becomes larger. Penalty factor has less drastic influence on error, only for values close to zero the error slightly grows but for almost all λ_1 there was no training error. By increasing the penalty factor the learning

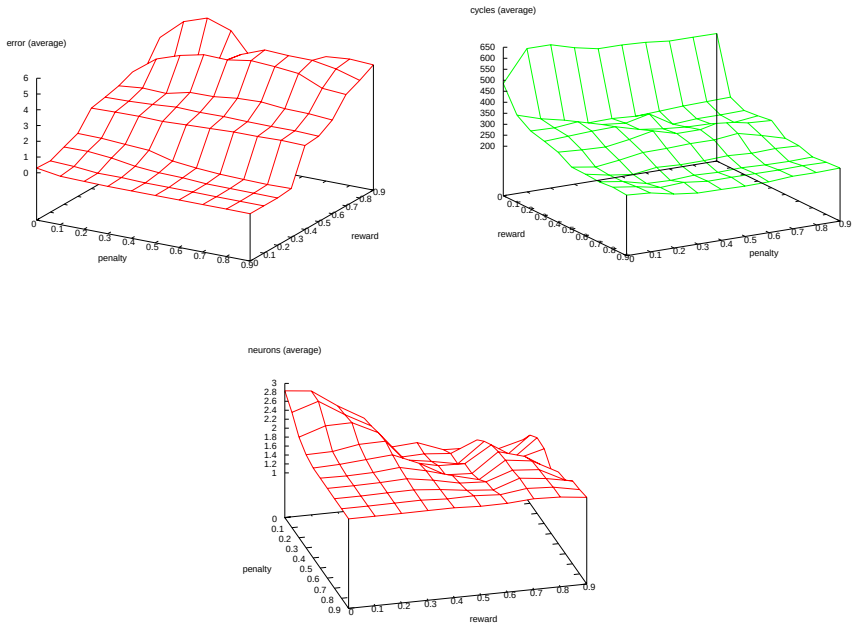

Fig. 2. Influence of penalty and reward factor on error (left), number of cycles (right) and number of neurons (bottom).

procedure becomes slower, more cycles are needed for convergence. Reward and penalty factors effect also the complexity of evolved network architecture. With small values of λ_1 and λ_2 the network ends with more neurons to achieve perfect solution. For values of penalty and reward $\lambda_1 \in [0.4-0.8]$ and $\lambda_2 \in [0.1-0.3]$ the network makes no mistakes, the convergence is fast and the network needs less neurons than the network without additional penalty and reward terms. Hence for further experiments $\lambda_1 = 0.5$ and $\lambda_2 = 0.2$ was taken.

Parity. Many models were presented to handle parity problem [1–3] but these models cannot be applied to other problems of similar complexity. Our network has easily learned all parity problems with dimensions less then 8, producing a very simple model in a small number of trials (initializations). For functions with higher dimension greater number of network initializations are needed to find the best solution (lowest k, with smallest number of clusters). In most cases weights of a single hidden unit converges to direction parallel to diagonal of n-dimension hypercube. Input vectors projected on this direction for parity problem gives well separated clusters of ones and zeros, thus this direction seems to be the desired solution. First hidden unit usually finds the largest cluster of ones. For example for the 6-dimensional parity problem projection of data on diagonal

direction gives the following order of labels:

$$01111110000000000000001\overline{1111111111111111111}0000000000000001111110$$

The line placed above the middle sequence of 1 shows the range of interval $[a, b]$ created by the first hidden node. Next node tries to find another direction and similar interval that contain remaining vectors from class 1. Projection on weights of the second and the third hidden neuron give

$$10000011111111110000000000\overline{1111111111}0000000000011111111111000001$$
$$01111100000000001111111110000\overline{11}0000011111111110000000000111110.$$

This is not k-separability solution as the projection directions were not aligned by weight sharing. With the increased dimension of the problem complexity of created network grows and the correct solution are harder to find. This is presented in Fig 3 where results of the training on parity problems are compared with results obtained for parity-like functions created from the parity function by perturbation of a few labels, and for the randomly generated labels with a probability $p(C = 1|\boldsymbol{x}) = 0.5$ for a given vector $\boldsymbol{x}$. Results were averaged over 100 randomly generated Boolean functions. While optimal solutions for some parity functions have not been found increasing the number of initializations should eventually find it.

In [6] sequential constructive methods based on computational geometry algorithms have been applied to the parity problems up to 9 bits. Computational time for all of them grows exponentially with the size of the problem. Most algorithms need a large number of neurons and may not generalize well. The sequential window learning algorithm was able to find models of minimal complexity. This algorithm also uses window-like neurons, but works only for binary data and employs learning algorithm based on computational learning techniques. While a detailed comparison of our approach with sequential constructive methods is certainly worthwhile the lack of software implementations for these methods makes such comparison a longer-term project.

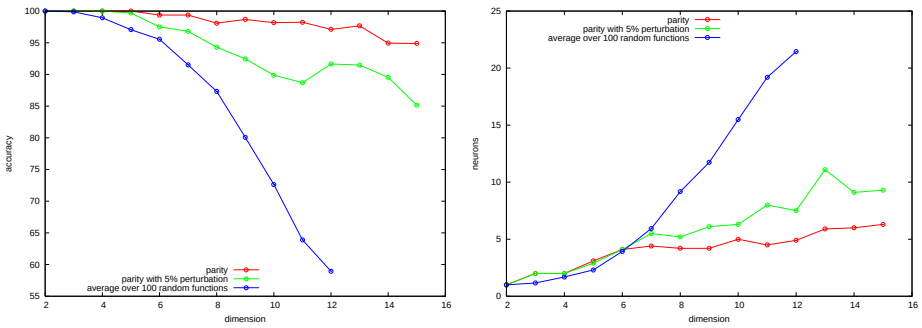

Fig. 3. Influence of dimensionality on the average training accuracy (left) and the average number of neurons (right) for some Boolean problems.

Monk 1 Problem. The three monk problems are artificial, small problems designed to test machine learning algorithms [13]. The task is to determine whether an object described by six symbolic features is "a monk" or not. The first problem defines "being a monk" as having the head shape = body shape, or jacket color = *red*, independent of the value of other features. There are 432 combinations of the 6 symbolic attributes, and for the first problem 124 cases were randomly selected for the training set. In [14] an MLP network solution of this problem requiring 4 neurons has been presented. Our network required only two window-like neurons to learn this problem and achieved 100% accuracy on the test. Among sequential constructive methods only the sequential window learning algorithm with Hamming clustering has correct solution, but this required conversion of 6 symbolic features to 15 binary ones.

Real World Problems. Creation of a separate network classifier for each class of data allows to handle real-world multi-class datasets. The classification test was performed on a few datasets obtained from UCI repository [15]. Three other well known classifiers were used for comparison of results. Table 3 presents averaged results over 10 runs of 10-fold cross-validation tests. Accuracy of our networks is listed in the last column of the table called "c3sep". Every node in the hidden layer was initialized 10 times and the data was normalized before training.

Table 1. Accuracy of the 10x10CV tests. Network parameters are $\lambda_1 = 0.5$, $\lambda_2 = 0.2$.

dataset	1-NN	Naive Bayes	SVM	c3sep
Appendicitis	81.3±1.5	85.3±1.0	86.5±0.3	85.3±1.0
Australian	78.0±0.2	80.0±0.3	85.0±0.1	86.3±0.6
Flag	50.1±1.1	41.1±1.1	51.1±1.1	53.6±1.8
Glass	68.2±1.7	47.4±1.7	66.7±0.9	61.1±1.3
Ionosphere	85.2±1.2	82.2±0.2	85.2±0.2	85.1±1.5
Iris	95.9±0.5	94.9±0.2	95.5±0.3	95.7±1.0
Pima-diabetes	70.5±0.5	75.2±0.5	70.3±1.0	76.3±0.4
Promoters	73.1±1.9	89.1±1.1	72.9±2.1	58.0±4.0
Sonar	86.8±1.8	67.8±1.2	84.2±1.1	77.9±2.4
Vowel	99.2±0.5	65.3±1.3	99.1±0.3	50.4±2.0
Wine	95.1±0.8	98.1±0.3	95.1±0.2	97.1±0.8

Table 2 shows the number of neurons that were created for a given datasets for each of the classes. For most datasets the network was able to find very simple solutions with only a few neurons, achieving quite good accuracy, comparable to much more complex systems. It should be stressed that the goal here is to find the simplest model, not the highest accuracy solution. The number of support vectors for SVM models with comparable accuracy is much higher.

Table 2. Average number of hidden neurons created in the 10x10CV classification test, and the number of support vectors used by SVM.

	support vectors	neurons (total)	average number of neurons per class
Appendicitis	32.1	4.2	2.2 2.0
Australian	207.2	8.8	4.5 4.3
Flag	315.2	26.7	5.1 6.1 4.0 2.1 2.7 3.6 1.1 1.6
Glass	295.8	14.0	1.4 3.9 1.0 2.8 1.9 2.8
Ionosphere	63.9	7.9	3.0 4.9
Iris	43.4	5.0	1.0 2.0 2.0
Pima-diabetes	365.3	9.1	4.2 4.8
Promotores	76.2	5.0	2.6 2.4
Sonar	109.7	8.5	4.5 4.0
Vowel	550.5	29.5	2.5 3.3 3.0 2.0 2.5 2.9 2.4 2.4 2.4 2.8 2.9
Wine	63.3	4.0	1.0 2.0 1.0

4 Discussion and Further Work

The approach presented here has been able to learn quite difficult Boolean functions using a very simple model. Other learning systems, such as SVMs, decision trees or similarity-based methods (including RBF networks) cannot deal with such problems. There is a widespread belief that neural networks and kernel classifiers, being universal approximators, should be able to learn any difficult problem. They may learn it but will not be able to generalize, turning themselves into look-up tables, because they do not use the correct underlying model to represent data.

The error function Eq. 5 with additional penalty and reward terms used to train the constructive network shows more advantages when dealing with complex logical problems. For many benchmark classification problems linear solutions are almost optimal and there is not much to be gained. Small computational costs of the constructive network training are a great advantage and allow for exploration of many models created from different initializations, enabling search for the simplest models (in the k-separability sense) that can solve the problem. Results obtained for the parity problem and the real-world data are encouraging, although the real power of solutions based on k-separability targets for learning should be seen only for data with inherent complex logics, such as those arising from the natural language processing and problems in bioinformatics (in preparation). The ability to solve such problems should open the door to many new applications.

Efficient learning of Boolean functions of high complexity (large k) is still a great challenge, although first steps towards this goal have been made here. Substitution of back-propagation learning by global minimization algorithms may lead to better results for complex functions with large number of bits. One disadvantage of the present approach based on constructive network architecture with modified error function is that the network is forced to create pure clusters only for vectors from class $c(x) = 1$, while for many Boolean problems a better solution may be obtained by looking for clusters of both classes simultaneously.

Another approach to improve learning is by additional transformation of input data before training, for example extracting new features using the kernel functions approach and adding them to the original dataset. The network trained on such data should be able to choose original or transformed features of the data. Many more ideas and applications are being currently explored.

Acknowledgments: We are grateful for the support by the Polish Committee for Scientific Research, research grant 2005-2007.

References

1. Iyoda, E., Nobuhara, H., Hirota, K.: A solution for the n-bit parity problem using a single translated multiplicative neuron. Neural Processing Letters **18** (2003) 233–238
2. Stork, D., Allen, J.: How to solve the n-bit parity problem with two hidden units. Neural Networks **5** (1992) 923–926
3. Wilamowski, B., Hunter, D.: Solving parity-n problems with feedforward neural network. In: Proc. of the Int. Joint Conf. on Neural Networks (IJCNN'03). Volume I., Portland, Oregon, IEEE Computer Society Press, Los Alamitos, CA (2003) 2546–2551
4. Sahami, M.: Learning non-linearly separable boolean functions with linear threshold unit trees and madaline-style networks. In: National Conference on Artificial Intelligence. (1993) 335–341
5. Liu, D., Hohil, M., Smith, S.: N-bit parity neural networks: new solutions based on linear programming. Neurocomputing **48** (2002) 477–488
6. Muselli, M.: Sequential constructive techniques. In Leondes, C., ed.: Optimization Techniques, vol. 2 of Neural Network Systems, Techniques and Applications. Academic Press, San Diego, CA (1998) 81–144
7. Duch, W.: k-separability. Lecture Notes in Computer Science **4131** (2006) 188–197
8. Duch, W., Jankowski, N.: Survey of neural transfer functions. Neural Computing Surveys (1999)
9. Bengio, Y., Delalleau, O., Roux, N.L.: The curse of highly variable functions for local kernel machines. Advances in Neural Information Processing Systems **18** (2006) 107–114
10. Bengio, Y., Monperrus, M., Larochelle, H.: Non-local estimation of manifold structure. Neural Computation **18** (2006) 2509–2528
11. Duch, W.: Towards comprehensive foundations of computational intelligence. In Duch, W., Mandziuk, J., eds.: Challenges for Computational Intelligence. Springer, Berlin, Heidelberg, New York (2007) in print
12. Duch, W., Jankowski, N.: Taxonomy of neural transfer functions. In: International Joint Conference on Neural Networks. Volume III., Como, Italy, IEEE Press (2000) 477–484
13. *et al.*, S.T.: The monk's problems: a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University (1991)
14. Duch, W., Adamczak, R., Grąbczewski, K.: A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. IEEE Transactions on Neural Networks **12** (2001) 277–306
15. Merz, C., Murphy, P.: UCI repository of machine learning databases (1998-2004) http://www.ics.uci.edu/∼mlearn/MLRepository.html.