Constructive Neural Network Algorithms that Solve Highly Non-Separable Problems

Marek Grochowski and Włodzisław Duch

In: Constructive Neural Networks, Springer Studies in Computational Intelligence
Vol. 258, pp. 49-70.
Eds. Franco L, Elizondo D.A, Jerez, J.M.

Abstract Learning from data with complex non-local relations and multimodal class distribution for widely used classification algorithms is still very hard. Even if accurate solution is found the resulting model may be too complex for a given data and will not generalize well. New types of learning algorithms are needed to extend capabilities of standard machine learning systems. Projection pursuit methods can avoid "curse of dimensionality" by discovering interesting structures in low-dimensional subspace. This paper introduces constructive neural architectures based on projection pursuit techniques that are able to discover simplest models of data with inherent highly complex logical structures.

Key words: Constructive neural networks, projection pursuit, non-separable problems, Boolean functions

1 Introduction

Most of the popular statistical and machine learning methods that rely solely on the assumption of local similarity between instances (equivalent to a smoothness prior) suffer from the curse of dimensionality[2]. When high-dimensional functions are not sufficiently smooth learning becomes very hard, unless extremly large number of training samples is provided. That leads to a dramatic cost increase of computations and creates complex models which are hard to interpret. Many data mining problems in bioinformatics, text analysis and other areas, have inherent complex logic. Searching for the simplest possible model capable of representing that kind of data is still a great challenge that has not been fully addressed.

For example, the n-bit parity problem has a very simple solution (one periodic function with a single parameter [3]), but popular kernel methods and algorithms

Marek Grochowski and Włodzisław Duch, Department of Informatics, Nicolaus Copernicus University, Toruń, Poland, e-mail: grochu@is.umk.pl; Google: W Duch

that depend only on similarity relations, or only on discrimination, have strong difficulties in learning this function. Linear methods fail completely, because this problem is highly non-separable. Gaussian-based kernels in SVMs use all training vectors as support vectors, because in case of parity function all points have closest neighbors from the opposite class. The nearest neighbor algorithms (with the numebr of neighbors smaller than $2n$) and the RBF networks have the same problem. For multilayer perceptrons convergence is almost impossible and requires many initiations to find accurate solution.

Special feedforward neural network architectures have been proposed to handle parity problems [16, 30, 31, 28, 21] but they are designed only for this special case and cannot be used for other Boolean functions, even very similar to parity. Learning systems are frequently tested on benchamrk datasets that are almost lienarly separable and relatively simple to handle, but without a strong prior knowledge it is very hard to find satisfactory solution for really complex problems. One can estimate how complex a given data is using the k-separability index introduced in [3]. Consider a dataset $\mathscr{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathscr{R}^d$, where each vector $\mathbf{x}_i$ belongs to one of the two classes.

Definition 1. Dataset $\mathscr{X}$ is called k-separable if a direction $\mathbf{w}$ exist such that all vectors projected on this direction $y_i = \mathbf{w}^T \mathbf{x}_i$ are clustered in k separated intervals, each containing instances from a single class only.

For example, datasets with two classes that can be separated by a single hyperplane have $k = 2$ and are thus 2-separable. XOR problem belongs to the 3-separable category, as projections have at least three clusters that contain even, odd and even instances (or odd, even and odd instances). n-bit parity problems are $n+1$-separable, because linear projection of binary strings exists that forms at least $n+1$ separated alternating clusters of vectors for odd and even cases. Please note that this is equivalent to a linear model with n parallel hyperplanes, or a nearest-prototype model in one dimension (along the line) with $n+1$ prototypes. This may be implemented as a Learning Vector Quantization (LVQ) model [19] with strong regularization. In both cases n linear parameters define direction $\mathbf{w}$, and n parameters define thresholds placed on the y line (in case of prototypes there are placed between thresholds, except for those on extreme left and extreme right, placed on the other side of the threshold in the same distance as the last prototype), so the whole model has $2n$ parameters.

It is obvious that complexity of data classification is proportional to the k-separability index, although for some datasets additional non-linear transformations are needed to avoid overlaps of projected clusters. For high values of k learning becomes very difficult and most classifiers, based on the Multi-Layer Perceptron (MLPs), Radial Basis Function (RBF) network, or Support Vector Machine (SVM) data models, as well as almost all other systems, are not able to discover simple data models. Linear projections are the simplest transformations with an easy interpretation. For many complicated situations proper linear mapping can discover interesting structures. Local and non-local distributions of data points can be clustered and discriminated by hyperplanes. Often a simple linear mapping exists that leaves only trivial non-linearities that may be separated using neurons that implement a window-like transfer function:

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

This function is suitable for learning all 3-separable data (including XOR). The number of such Boolean functions for 3 or more bits is much greater than of the linearly separable functions [3]. For data which is more than $k = 3$ separable this will not give an optimal solution, but it will still be simpler than the solution constructed using hyperplanes. There are many advantages of using window-type functions in neural networks, especially in difficult, highly non-separable classification problems [7]. One of the most interesting learning algorithms in the field of learning Boolean functions is the constructive neural network with Sequential Window Learning (SWL), an algorithm described by Muselli [26]. This network also uses window-like transfer function, and in comparison with other constructive methods [26] outperforms similar methods with threshold neurons, leading to models with lower complexity, higher speed and better generalization [12]. SWL works only for binary data and therefore some pre-processing is needed to use it for different kind of problems.

In the next section constructive network is presented that uses window-like transfer functions to distinguish clusters created by linear projections. A lot of methods that search for optimal and most informative linear transformations have been developed. Projection pursuit is a branch of statistical methods that search for interesting data transformations by maximizing some “index of interest” [18, 10]. Constructive network described in section 3 use projection pursuit methodology for construction of an accurate neural architecture for solving complex problems.

2 Constructive 3-separability model (c3sep)

With growing k-separability index problems quickly become intractable for general classification algorithms. Although some problems with high k 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. In feedforward multilayer neural networks hidden layers represents some data transformation which should lead to linear separation of samples in the output layer. For highly-nonseparable data this transformation in very hard to find.

In case of backpropagation procedure, when all network weights are adjusted in each iteration, convergence to the optimal solution is almost impossible. The final MLP model gives no information about the structure of the problem, representing data structures in completly distributed way. Using constructive methods, a single node can be trained separately, providing a partial solution. As a result each network node represents a chunk of knowledge about the whole problem, focusing on different subsets of data and facilitating interpretation of the data structure.

Window-type transfer functions

In the brain neurons are organized in cortical column microcircuits [22] that resonate with certain frequencies whenever they receive specific signals. Threshold neurons split input space in two disjoint regions, with hyperplane defined by the **w** direction. For highly non-separable data searching for linear separation is useless, while finding interesting clusters for projected data, corresponding to an active microcuircuit, is more liekely. Therefore network nodes should implement a window-like transfer functions (Eq. 1) that solve 3-separable problems by combination of projection and clustering, separating some (preferably large) number of instances from a single class in the $[a,b]$ interval. This simple transformation may handle not only local neigborhoods, as Gaussian functions in SVM kernels or RBF networks do, but also non-local distributions of data that typically appear in the Boolean problems. For example, in the n-bit parity problem projection on the $[1,1..1]$ direction creates several large clusters with vectors that contain fixed number of 1 bits.

Optimization methods based on gradient descent used in the error backpropagation algorithm require continuous and smooth functions, therefore soft windowed-type functions should be used in the training phase. Good candidate functions include a combination of two sigmoidal functions:

$$M(\mathbf{x};\mathbf{w},a,b,\beta) = \sigma(\beta(\mathbf{wx}-a)) - \sigma(\beta(\mathbf{wx}-b)) \tag{2}$$

For $a > b$ an equivalent product form, called bicentral function [5], is:

$$M(\mathbf{x};\mathbf{w},a,b,\beta) = \sigma(\beta(\mathbf{wx}-a))(1-\sigma(\beta(\mathbf{wx}-b)) \quad . \tag{3}$$

Parameter β controls slope of the sigmoid functions and can be adjusted during training together with weights **w** and biases a and b. Bicentral function (3) has values in the range $[0,1]$, while function (2) for $b < a$ may become negative, giving values in the $[-1,+1]$ range. This property may be useful in constructive networks for “unlearning” instances, misclassified by previous hidden nodes. Another interesting window-type function is:

$$M(\mathbf{x};\mathbf{w},a,b,\beta) = \frac{1}{2}\big(1-\tanh(\beta(\mathbf{wx}-a))\tanh(\beta(\mathbf{wx}-b))\big) \quad . \tag{4}$$

This function has one interesting feature: for points $\mathbf{wx} = a$ or $\mathbf{wx} = b$ and for any value of slope β it is equal to $1/2$. By setting large value of β hard-window type function (1) is obtained

$$M(\mathbf{x};\mathbf{w},a,b,\beta) \overset{\beta\to\infty}{\longrightarrow} \tilde{M}(\mathbf{x};\mathbf{w},a',b') \tag{5}$$

where for function (4) boundaries of the $[a,b]$ interval do not change ($a = a'$ and $b = b'$), while for the bicentral function (3) value of β has influence on the interval boundaries, so for $\beta \to \infty$ they are different than $[a,b]$. Another way to achieve sharp decision boundaries is by introduction of an additional threshold function and parameter:

$$\tilde{M}(\mathbf{x};\mathbf{w},a',b') = \mathrm{sgn}\left(M(\mathbf{x};\mathbf{w},a,b,\beta) - \theta\right) \ . \tag{6}$$

Many other types of transfer functions can be used for practical realization of 3-separable models. For detailed taxonomy of neural transfer functions see [7, 8].

Modified error function

Consider a dataset $\mathscr{X} \subset \mathscr{R}^d$, where each vector $\mathbf{x} \in \mathscr{X}$ belongs to one of the two classes $c(\mathbf{x}) \in \{0,1\}$. To solve this classification problem neural network should minimize an error measure:

$$E(\mathscr{X};\Gamma) = E_{\mathbf{x}}||y(\mathbf{x};\Gamma) - c(\mathbf{x}))|| \tag{7}$$

where $y(\mathbf{x};\Gamma)$ is the network output and Γ denotes a set of all parameters that need to be adjusted during training (weights, biases, etc.). The expectation value is calculated over all training vectors using the mean square error, or cross entropy, or other norms suitable for error measures. However, in constructive networks nodes may be trained separately, one at a time, and a partial solution in form of pure clusters for some range of $[a_i,b_i]$ output values created by each $M(\mathbf{x};\Gamma_i)$ node are used to improve the network function. To evaluate a usefulness of a new node $M(\mathbf{x};\Gamma_i)$ for the network $y(\mathbf{x};\Gamma)$, where Γ represents all parameters, including Γ_i, an extra term is added to the standard error measure:

$$E(\mathscr{X};\Gamma;a,b,\lambda) = E_{\mathbf{x}}||y(\mathbf{x};\Gamma) - c(\mathbf{x}))|| + \lambda_i E_{M\in[a_i,b_i]}||M(\mathbf{x};\Gamma_i) - c(\mathbf{x}))|| \tag{8}$$

where λ_i controls the tradeoff between the covering and the quality of solution after the new $M(\mathbf{x};\Gamma_i)$ hidden node is added. For nodes implementing $\mathbf{wx}$ projections (or any other functions with outputs restricted to $[a_i,b_i]$ interval) largest pure cluster will give the lowest contribution to the error, lowering the first term while keeping the second one equal to zero. If such cluster is rather small it may be worthwhile to create a slightly bigger one, but not quite pure, to lower the first term at the expense

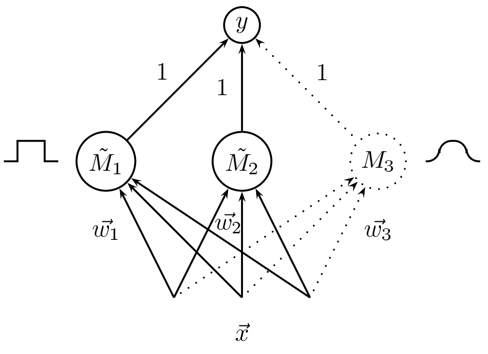

Fig. 1 Example of the *c3sep* network with three hidden neurons. Only parameters of the last node are adjusted during training (dotted line), the first and the second node have been frozen, with large value of β used to obtain sharp interval boundaries.

of the second. Usually a single λ_i parameter is taken for all nodes, although each parameter could be individually optimized to reduce the number of misclassifications.

The current version of the *c3sep* constructive network assumes binary $0/1$ class labeles, and uses the standard mean square error (MSE) measure with two additional terms:

$$\begin{aligned} E(\mathbf{x};\Gamma,\lambda_1,\lambda_2) = &\frac{1}{2}\sum_{\mathbf{x}}(y(\mathbf{x};\Gamma)-c(\mathbf{x}))^2 + \\ &+\lambda_1\sum_{\mathbf{x}}(1-c(\mathbf{x}))y(\mathbf{x};\Gamma) - \lambda_2\sum_{\mathbf{x}}c(\mathbf{x})y(\mathbf{x};\Gamma) \end{aligned} \tag{9}$$

The term scaled by λ_1 represents additional the penalty for “unclean” clusters, increasing the total error for vectors from class 0 that falls into at least one interval created by hidden nodes. The term scaled by λ_2 represents reward for large clusters, decreasing the value of total error for every vector that belongs to class 1 and was correctly placed inside created clusters.

The *c3sep* architecture and training

The output of the *c3sep* network is given by:

$$y(\mathbf{x};\Gamma) = \sigma\left(\sum_i M(\mathbf{x};\Gamma_i) - \theta\right) \tag{10}$$

where $\Gamma_i = \{\mathbf{w}_i, a_i, b_i, \beta_i\} \subset \Gamma$ denote subset of parameters of i-th node. All connections between hidden and output layer are fixed with strength 1, although in the final step they could be used as a linear layer for additional improvement of the network. One can use linear node as an output, but in practice sigmoidal function provides better convergence. The architecture of this network is shown in Fig. 1. Each hidden node tries to separate a large group of vectors that belong to class $c = 1$. Learning procedure starts with an empty hidden layer. In every phase of the training one new window-type unit is added and trained by minimization of the error function (9). Weights of each node are initialized with small random values before training. Initial values for biases a and b can be set to

$$a = (\mathbf{wx})_{min} + \frac{1}{3}|(\mathbf{wx})_{max} - (\mathbf{wx})_{min}|$$

$$b = (\mathbf{wx})_{min} + \frac{2}{3}|(\mathbf{wx})_{max} - (\mathbf{wx})_{min}|$$

In most cases this should provide a good starting point for optimization with gradient based methods.

Network construction proceeds in a greedy manner. First node is trained to separate as large group of class 1 vectors as possible. After convergence is reached the slope of transfer function is set to a large value to obtain hard-windowed function, and the weights of this neuron are kept frozen during further learning. Samples from class 1 correctly handled by the network do not contribute to the error, and can be removed from the training data to further speed up learning (however, leaving them may stabilize learning, giving a chance to form more large clusters). After that, the next node is added and the training repeated on the remaining data, until all vectors are correctly handled. To avoid overfitting, one may use pruning techniques, as it is done in the decision trees. The network construction should be stopped when the number of cases correctly classified by a new node becomes too small, or when the crossvalidation tests show that adding such node will decrease generality.

Experiments

2.1 Boolean functions

The *c3sep* network has been tested on several types of problems. Figures 2 and 3 shows results of learning on Boolean problems with systematically increasing complexity. Results are compared with a few neural constructive algorithms designed to deal with Boolean functions. All these algorithms may be viewed as a realization of general sequential constructive method [26] (this method is briefly described in subsection 3), and differ by strategy of searching for the best hidden nodes. Irregular Partitioning algorithm [23] uses threshold perceptrons optimized with linear programing. Carve [32] is trained by the convex hull traversing algorithm. Oil Spot algorithm [24] searches for connected subgraphs and proper edge orientation in hypercube graph. Sequential Window Learning method [25] uses window transfer functions for which weights are obtained from solution of a system of algebraic equations. Target Switch algorithm [33] use traditional perceptron learning.

Fig. 2 presents results of learning of constructive methods applied to the parity problems from 2 bits to 10 bits, and Fig. 3 shows the results of learning randomly selected Boolean functions, where labels for each string of bits have been drawn with equal probability $P(C(\mathbf{x}) = 1) = 0.5$. The same random function was used to train all algorithms. These figures show the size of networks constructed by a given algorithm, and the time needed for learning until a given problem has been solved without mistakes.

The *c3sep* network avoids small clusters increasing generalization, and uses stochastic gradient algorithm, that avoids local minima through multistarts, and thus leads to small errors in some runs. Values of the training error are placed in corresponding points of Fig. 2 and Fig. 3. The n-dimensional parity problem can be solved by a two-layer neural network with n threshold neurons or $(n+1)/2$ window-like neurons in the hidden layer[3]. Sequential window learning and irregular partitioning algorithms were able do obtain optimal solution for all dimensions.

Learning of random Boolean functions is much more difficult, and upper bound for the number of neurons needed for solving of this kind of functions is not known. This purpose of these tests is to check the ability of each algorithm to discover simple models of complex logical functions. Algorithms capable of exact learning of every example by creating separate node for single vectors are rarely useful as they will overfit the data. Therefore the ability to find simple, but approximate, solutions is very useful. One should expect that such approximate models should be more robust than perfect models if the training is carried on slightly different subset of examples for a given Boolean function.

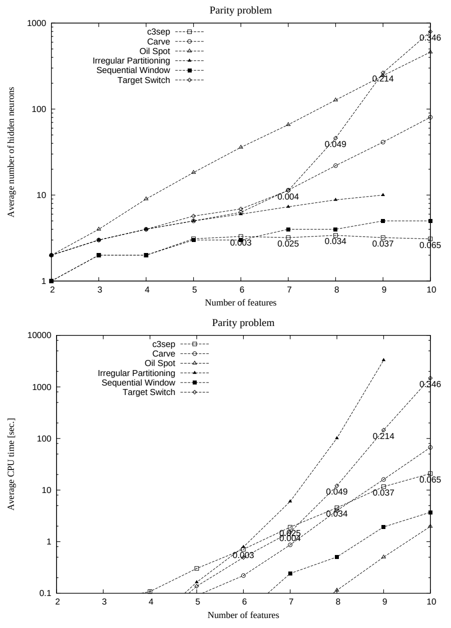

Fig. 2 Number of hidden units created, and time consumed during learning of parity problems. Each result is averaged over 10 trials.

Irregular partitioning produces small networks, but the training time is very high, while on the other hand the fastest methods (Oil Spot) needs many neurons. Sequential window learning gave solutions with a small number of neurons and rather low computational cost. The *c3sep* network was able to create smallest architectures, but the average times of computations are somewhow longer than needed by most other algorithms. This network provides near optimal solution, as not all patterns were correctly classified.

2.2 Real world problems

Tables 1 and 2 present results of a generalization tests for a few benchmark datasets from the UCI repository [1]. The Iris dataset is perhaps the most widely used simple problem, with 3 types of Iris flowers described by 4 real valued attributes. The Glass identification problem has 9 real valued features with patterns divided into float-processed and non float-processed pieces of glass. United States congressional voting record database, denoted here as Voting0 dataset, contains 12 features that record decisions of congressmans who belong to a democratic or republican party. The Voting1 dataset has been obtained from the Voting0 by removing the most informative feature. Each input can assume 3 values: yea, nay or missing.

Some algorithms used for comparison work only for binary features, therefore their application requires additional pre-processing of data vectors. Real valued fea-

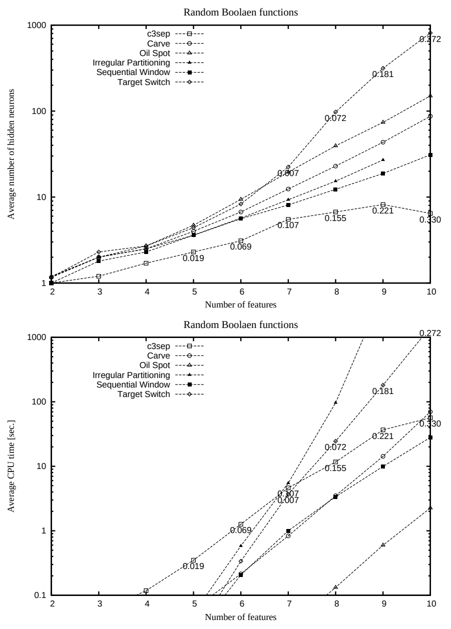

Fig. 3 Number of hidden units created and time consumed during learning of random Boolean functions. Each result is averaged over 10 trials.

tures have been transformed to binary features by employing Gray coding [27]. Resulting Iris and Glass dataset in binary representation have 22 and 79 features, respectively. For the three-valued input of Voting dataset a separate binary feature has been associated with the presence of each symbolic value, resulting in 48 binary features for Voting0, and 45 for Voting1 datasets.

Algorithms that can handle real features have been applied to both original data and binarized data (Tab. 1). Although dimensionality of binary data is higher (and thus more adaptive parameters are used by standard MLP networks and other algorithms), results of most methods on binary data are significantly worse, particularly in the case of Iris and Glass where all features in the original data are real valued.

In all these tests *c3sep* network gave very good accuracy with low variance, better on statistically equivalent to the best solutions, with a very small number of neurons created in the hidden layer. The ability to solve complex problems in an approximate way is evidently helpful also for relatively simple data used here, showing the universality of constructive *c3sep* networks.

Table 1 30x3 CV test accuracy.

	Iris	**Glass**	**Voting0**	**Voting1**
Carve	90.18 ± 1.58	74.17 ± 3.28	93.24 ± 1.00	87.45 ± 1.37
Irregular Partitioning	90.98 ± 2.29	72.29 ± 4.39	93.41 ± 1.13	86.96 ± 1.43
Target Switch	65.45 ± 5.05	46.76 ± 0.91	94.64 ± 0.63	88.13 ± 1.47
c3sep	95.40 ± 1.30	70.68 ± 2.97	94.38 ± 0.72	90.42 ± 1.15
binary features				
Oil Spot	75.16 ± 2.86	66.05 ± 2.41	90.93 ± 0.90	86.68 ± 1.47
Carve	71.84 ± 3.46	62.08 ± 4.55	91.79 ± 1.22	86.77 ± 1.43
Irregular Partitioning	75.53 ± 3.20	62.38 ± 3.66	92.73 ± 1.19	86.79 ± 2.20
Target Switch	84.93 ± 3.28	71.69 ± 3.42	94.66 ± 0.69	88.36 ± 0.98
c3sep	75.58 ± 6.15	60.92 ± 4.47	94.50 ± 0.89	89.78 ± 1.26

Table 2 Average number of hidden neurons generated during 30x3 CV test.

	Iris	**Glass**	**Voting0**	**Voting1**
Carve	5.72 ± 0.46	7.00 ± 0.50	4.99 ± 0.39	8.34 ± 0.45
Irregular Partitioning	5.49 ± 0.53	4.69 ± 0.26	2.04 ± 0.21	3.48 ± 0.30
Target Switch	22.76 ± 2.17	55.49 ± 2.38	3.69 ± 0.29	9.22 ± 0.85
c3sep	3.00 ± 0.00	1.14 ± 0.26	1.00 ± 0.00	1.02 ± 0.12
binary features				
Oil Spot	27.78 ± 1.41	21.54 ± 1.80	22.76 ± 1.39	37.32 ± 2.32
Carve	8.02 ± 0.52	6.79 ± 0.26	5.56 ± 0.32	8.59 ± 0.46
Irregular Partitioning	3.00 ± 0.00	1.00 ± 0.00	0.99 ± 0.06	2.50 ± 0.30
Target Switch	3.07 ± 0.14	1.72 ± 0.25	3.20 ± 0.26	7.46 ± 0.48
c3sep	3.30 ± 0.35	1.03 ± 0.10	1.00 ± 0.00	1.00 ± 0.00

3 Projection Pursuit Constructive Neural Network

Projection pursuit (PP) is a generic name given to all algorithms that search for the most “interesting” linear projections of multidimensional data, maximizing (or minimizing) some objective functions or indices [11, 10]. Many projection pursuit indices may be defined to characterize different aspects or structures that the data may contain. Modern statistical dimensionality reduction approaches, such as the principal component analysis (PCA), Fisher’s discriminant analysis (FDA) or independent component analysis (ICA) may be seen as special cases of projection pursuit approach. Additional directions may be generated in the space orthogonalized to the already found directions.

PP indices may be introduced both for unsupervised and for supervised learning. By working in a low-dimensional space based on linear projections projection pursuit methods are able to avoid the “curse of dimensionality” caused by the fact that high-dimensional space is mostly empty [15]. In this way noisy and non-informative variables may be ignored. In contrast to most similarity-based methods that optimize metric functions to capture local clusters, projection pursuit may discover also non-local structures. Not only global, but also local extrema of the PP index are of interest and may help to discover interesting data structures.

A large class of PP constructive networks may be defined, where each hidden node is trained by optimization of some projection pursuit index. In essence the hidden layer defines a transformation of data to low dimensional space based on sequence of projections. This transformation is then followed by linear discrimination in the output layer. PCA, FDA or ICA networks are equivalent to linear discrimination on pre-processed suitable components. In the next section more interesting index, in the spirit of k-separability, is defined.

The QPC Projection Index

Consider a dataset $\mathscr{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathscr{R}^d$, where each vector $\mathbf{x}_i$ belongs to one of the k different classes. Let $\mathscr{C}_\mathbf{x}$ denote the set of all vectors that have the same label as $\mathbf{x}$. The following index achieves maximum value for projections on the direction $\mathbf{w}$ that groups all vectors from class $\mathscr{C}_\mathbf{x}$ into a compact cluster separated from vectors that belong to other classes:

$$Q(\mathbf{x};\mathbf{w}) = A^+ \sum_{\mathbf{x}_k \in \mathscr{C}_\mathbf{x}} G\left(\mathbf{w}^T(\mathbf{x}-\mathbf{x}_k)\right) - A^- \sum_{\mathbf{x}_k \notin \mathscr{C}_\mathbf{x}} G\left(\mathbf{w}^T(\mathbf{x}-\mathbf{x}_k)\right) \tag{11}$$

where $G(x)$ is a function with localized support and maximum in $x = 0$, for example a Gaussian function. The first term in $Q(\mathbf{x};\mathbf{w})$ function is large if all vectors from class $\mathscr{C}_\mathbf{x}$ are placed close to $\mathbf{x}$ after the projection on direction defined by $\mathbf{w}$, indicating how compact and how large is this cluster of vectors. The second term depends on distance beetwen $\mathbf{x}$ and all patterns that do not belong to class $\mathscr{C}_\mathbf{x}$, therefore it represents penalty for placing vector $\mathbf{x}$ too close to the vectors from opposite classes. The Quality of Projected Clusters (QPC) index is defined as an average of the $Q(\mathbf{x};\mathbf{w})$ for all vectors:

$$QPC(\mathbf{w}) = \frac{1}{n}\sum_{\mathbf{x}\in\mathscr{X}} Q(\mathbf{x};\mathbf{w}) \;, \tag{12}$$

providing a leave-one-out estimator that measures quality of clusters projected on $\mathbf{w}$ direction. This index achieves maximum value for projections $\mathbf{w}$ that create small number of pure, compact and well separated clusters. For linearly separable problems function $QPC(\mathbf{w})$ achieves maximum for projections $\mathbf{wx}$ that create two well-separated pure clusters of vectors. In case of k-separable dataset maximization of QPC index leads to a projection with k separated clusters. Thus optimization of QPC should discover k-separable solutions if they exist, otherwise directions

Parameters A^+, A^- control influence of each term in Eq. (11). If A^- is large strong separation between classes is enforced, while large A^+ impacts mostly compactness and purity of clusters. For example, by setting $A^+ = p(\mathscr{C}_\mathbf{x})$ and $A^- = 1 - p(\mathscr{C}_\mathbf{x})$ (where $p(\mathscr{C}_\mathbf{x})$ is the *a priori* class probability), projection index is balanced in respect to the size of classes. If in addition $G(x)$ is normalized, such that $G(0) = 1$, then the upper limit of QPC index is 1 and it occurs only when all vectors from the same class after projection are placed in a single very narrow cluster and the gap beetwen each cluster is greater than the range of $G(\mathbf{x})$ function. All bell-shaped functions that achieve maximum value for $x = 0$ and vanish for $x \to \pm\infty$ are suitable for $G(x)$, including Gaussian, bicentral functions Eq. (3) and Eq. (2), or an inverse quartic function:

$$G(x) = \frac{1}{1+(bx)^4} \tag{13}$$

where parameter b controls the width of $G(x)$.

These functions are continuous and thus may be used in gradient-based methods. Iterative gradient optimization procedures applied to functions with multiple local minima do no guarantee that an optimal solution will be found, and may converge slowly. Direct calculation of the QPC index (12) requires $O(n^2)$ operations (after projection distances beetwen all pairs of vectors are computed), as in the nearest neighbor methods. For large datasets this may be excessive.To overcome this problem various “editing techniques”, or instance selection algorithms developed for the nearest neighbor methods may be used [17, 13, 29]. By sorting projected vectors and restricting computations of the sum in Eq. (11) only to vectors $\mathbf{x}_i$ with $G(\mathbf{w}(\mathbf{x}-\mathbf{x}_i)) > \varepsilon$ computational time is easily decreased to $O(n\log n)$. Further improvements in speed may be achieved if the sum in Eq. (12) is restricted only to a few centers of projected clusters $\mathbf{t}_i$. This may be done after projection $\mathbf{w}$ stabilizes, as at the beginning of the training the number of clusters in the projection is not known without some prior knowledge about the problem. For k-separable datasets k centers are sufficient and the cost of computing QPC index drops to $O(kn)$. Gradient descent methods may be replaced by more sophisticated approaches [14, 20]), although in practice multistart gradient methods have been quite effective in searching for interesting projections. It is worth to notice that although global extrema of QPC index give most valuable projections, suboptimal solutions may also provide useful insight into the structure of data.

First QPC direction

Figure 4 presents projections for 4 very different kinds of datasets: Wine, Monk1, 10-bit Parity and Concentric Rings. All projections were obtained taking quartic function (13) for $G(x)$, with $b = 3$, and using simple gradient descent maximization initialized 10 times, selecting after a few iterations the most promising solution that is trained until convergence. Values of weights and the value of $QPC(\mathbf{w})$ are shown in the corresponding figures. Positions of projected vectors on the line are shown each class below the projection line. Smoothed histograms for these projections may be normalized and taken as estimations of class conditionals $p(x|C)$, from which posterior probabilities $p(C|x) = p(x|C)p(C)/p(x)$ are easily calculated.

The first two datasets are taken from the UCI repository [1]. The Wine data consist of 178 vectors, 13 continuous features and 3 classes. It can be classified using a single linear projection that gives 3 groups of vectors (one for each class). The weight for “flavanoids” feature dominates and is almost sufficient to separate all 3 classes. Monk 1 is an artificial datasets [1], with 6 symbolic features and two classes, defined by two simple rules: given object is “a monk” if “the head shape” (first attribute) = “body shape” (second attribute) or “jacket color” (fifth attribute) = red. Direction generated by maximization of the QPC index produces large cluster of vectors in the middle of the projection. First two coefficients are large and equal, others are essentially zero. This corresponds to the first rule, but the second rule cannot be captured by the same projection. To separate the remaining cases a second projection is needed (see below). These logical rules have also been extracted using a special version of MLP network [5].

The 10 bit parity is an example of a hard Boolean problem, where 1024 samples are divided into even and odd binary strings. This problem is 11-separable, with a maximum value of projection index obtained for diagonal direction in the 10 dimensional hypercube, therefore all weights have the same value. Although a perfect solution using a single projection has been found clusters at the extreme left and right of the projection are quite small, therefore finding another direction that puts these vectors in larger clusters may be useful. Convergence of MLP or RBF networks for such complex data is quite unlikely, but also standard SVM approaches fail completely if crossvalidation tests are performed.

The final dataset (Concentric Rings) contains 800 samples distributed in 4 classes, each with 200 samples defined by 4 continuous features. Only the first and the second feature is relevant, vectors belonging to the same class are located inside one of the 4 concentric rings. The last two noisy features are uniformly distributed random numbers. For this dataset the best projection that maximizes the QPC index reduces influence of noisy features, with weights for dimensions 3 and 4 close to zero. This shows that the QPC index may be used for feature selection, but also that linear projections have limited power: a complicated solution requiring many projections at different angles to delineate the rings is needed. Of course a much simpler network using localized functions will solve this problem more accurately. The need for networks with different types of transfer functions [7, 9] has been stressed some time ago, but still there are no programs capable of finding the simplest data models in all cases.

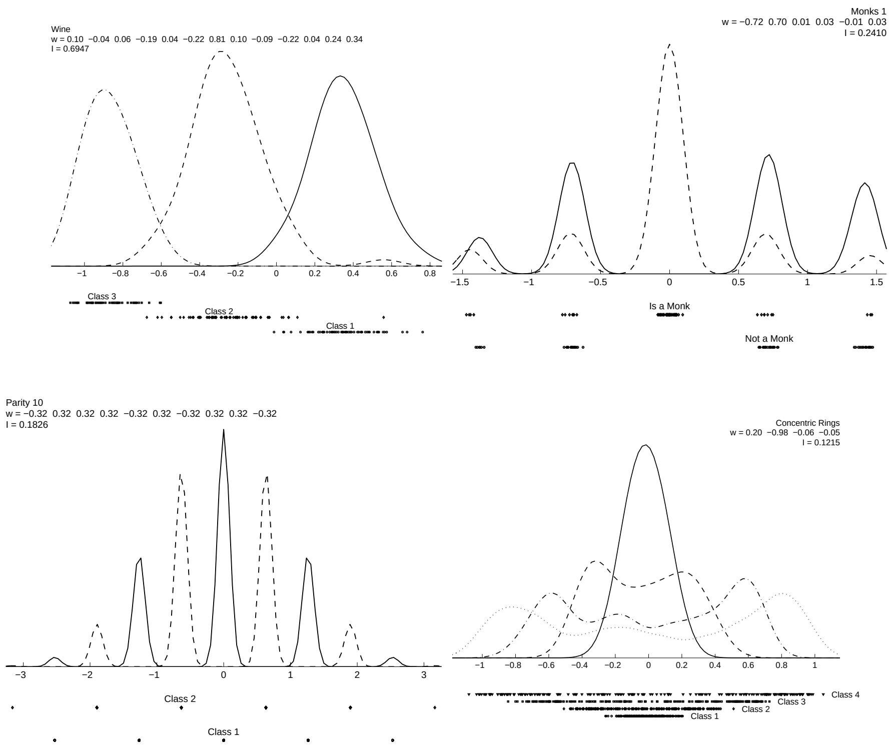

Fig. 4 Examples of four projections found by maximization of the QPC index using gradient descent for the Wine data (top-left), the Monk1 problem (top-right), the 10-bit Parity (bottom-left) and the noisy Concentric Rings (bottom-right).

Second QPC direction

For complex problems usually more than one projection is required. Using QPC index searching for additional interesting projections can be realized in several ways. Sequence of unique directions may be generated applying repeatedly QPC optimization in the subspace orthogonal to all directions found earlier. Another possible approach is to focus on subsets of vectors with poor separation and search for another direction only for overlapping clusters until separation is attained. The third possibility is to search for the next linear projection with additional penalty term that will punish solutions similar to those found earlier:

$$QPC(\mathbf{w};\mathbf{w}_1) = QPC(\mathbf{w}) - \lambda f(\mathbf{w},\mathbf{w}_1) \tag{14}$$

The value of $f(\mathbf{w},\mathbf{w}_1)$ should be large if the current direction $\mathbf{w}$ is close to the previous direction $\mathbf{w}_1$. For example, some power of the scalar product between these directions may be used: $f(\mathbf{w},\mathbf{w}_1) = (\mathbf{w}_1{}^T \cdot \mathbf{w})^2$. Parameter λ scales the importance of enforcing this condition during the optimization process.

Scatterplots of data vectors projected on two directions may be used for visualization. Figure 5 presents such scatterplots for the four datasets used in the previous section. The second direction $\mathbf{w}$, found by gradient descent optimization of function (14) with $\lambda = 0.5$, is used for the horizontal axis. The final weights of the second direction, value of the projection index $QPC(\mathbf{w})$, and the inner product of $\mathbf{w}_1$ and $\mathbf{w}$ are shown in the corresponding figures.

For the Wine problem first projection was able to separate almost perfectly all three classes. Second projection (Fig. 5) gives additional insight into the structure of this data, leading to a better separation of vectors placed near decision boundary.

Two-dimensional projection of Monk1 data shows separate and compact clusters. The 5th feature (which forms the second rule describing this dataset: if it is 1 then object is a Monk) has significant value, and all unimportant features have weights equal almost zero, allowing for simple extraction of correct logical rules.

In case of the 10-bit parity problem each diagonal direction of a hypercube representing Boolean function gives a good solution with large cluster in the center. Two such orthogonal directions have been found, projecting each data vector into large pure cluster, either in the first or in the second dimension. In particular small, one or two-vector clusters at the extreme ends of the first projection belong to the largest clusters in the second direction, ensuring good generalization in this two-dimensional space using naive Bayes estimation of classification probabilities.

Results for the noisy Concentric Rings dataset show that maximization of the QPC index has caused vanishing of noisy and uninformative features, and has been able to discover two-dimensional relations hidden in this data. Although linear projections in two directions cannot separate this data, such dimensionality reduction is sufficient for any similarity-based method, for example the nearest neighbor method, to perfectly solve this problem.

A single projection allows for estimation and drawing class-conditional and posterior probabilities, but may be not sufficient for optimal classification. Projections on 2 or 3 dimensions allow for visualization of scatterograms, showing data structures hidden in the high-dimensional distributions, suggesting how to handle the problem in the simplest way: adding linear output layer (Wine), employing localized functions, decision trees or covering algorithms, using intervals (parity) or naive Bayes, or using the nearest neighbor rule (Concentric Rings). If this is not sufficient

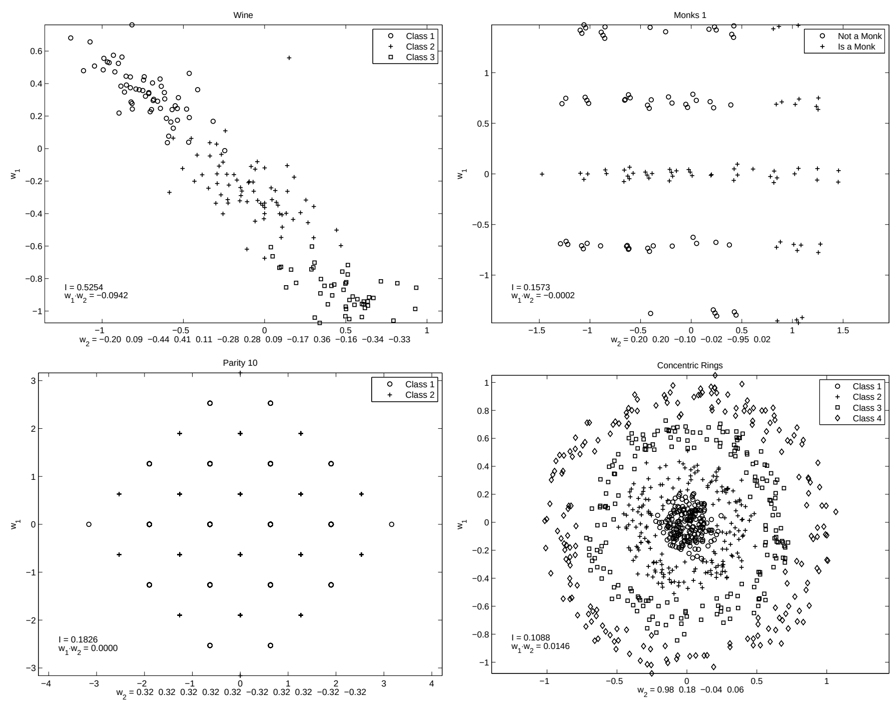

Fig. 5 Scatterplots created by projection on two QCP directions for the Wine and Monk1 data (top-left/right), 10-bit parity and the noisy Concentric Rings data (bottom-left/right).

more projections should be used as a pre-processing for final classification, trying different approaches in a meta-learning scheme [6].

Coefficients of the projection vectors may be used directly for feature ranking/selection, because maximization of the QPC index gives negligible weights to noisy or insignificant features, while important attributes have distinctly larger values. This method might be used to improve learning for many machine learning models sensitive to feature weighting, such as all similarity-based methods. Interesting projections may also be used to initialize weights in various neural network architectures.

Constructive Neural Network Based on the QPC Index

Projection pursuit methods in a natural way may be used for constructive neural networks learning, where each hidden node coresponds to a linear mapping obtained by optimization of a projection index. To build a neural network architecture using QPC index general sequential constructive method may be used [26]. For the two-class problems this method is described as follows:

1. start learning with an empty hidden layer;
2. if there are some misclassfied vectors *do*:
3. add a new node;
4. train the node to obtain a partial classiffier;
5. remove all vectors for which the current node outputs $+1$;
6. *enddo*.

A partial classiffier is a node with output $+1$ for at least one vector from one of the classes, and -1 for all vectors from the opposite classes. After a finite number of iterations this procedure leads to a construction of neural network that classifies all training vectors (unless there are conflicting cases, i.e. identical vectors with different labels, that should be removed). Weights in the output layer do not take part in the learning phase and their values can be determined from a simple algebraic equation, assigning the largest weight to the node created first, and progressively smaller weights to subsequently created nodes, for example:

$$u_0 = \sum_{j=1}^{h} u_j + d_{h+1} \quad , \qquad u_j = d_j 2^{h-j} \quad \text{for} \quad j = 1, \ldots, h \tag{15}$$

where h is the number of hidden neurons, $d_i = \{-1, +1\}$ denotes label for which i-th hidden node gives output $+1$ and $d_{h+1} = d_h$.

The sequential constructive method critically depends on construction of good partial classifier. A method to create it is described below. Consider a node M implementing the following function:

$$M(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad |G(\mathbf{w}(\mathbf{x}-\mathbf{t})) - \theta| \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{16}$$

where the weights $\mathbf{w}$ are obtained by maximization of the QPC index, θ is a threshold parameter which determins the window width, and $\mathbf{t}$ is the center of a cluster of vectors projected on $\mathbf{w}$, estimated by:

$$\mathbf{t} = \arg\max_{\mathbf{x}\in\mathscr{X}} Q(\mathbf{w}, \mathbf{x}) \tag{17}$$

If the direction $\mathbf{w}$ corresponds to the maximum of QPC index then $\mathbf{t}$ should be at the center of a large, clean and well separated cluster. Thus the node (16) splits input space into two disjoint subspaces, with output $+1$ for each vector that belongs to the cluster, and -1 for all other vectors. Large, clean and well-separated clusters may be achieved by maximization of the function $Q(\mathbf{t}; \mathbf{w})$ with respect to weights $\mathbf{w}$ and cluster center $\mathbf{t}$, or by minimization of an error function:

$$E(\mathbf{x}) = E_{\mathbf{x}} ||G(\mathbf{w}(\mathbf{x}-\mathbf{t})) - \delta(c_{\mathbf{x}}, c_{\mathbf{t}})|| \tag{18}$$

where $\delta(c_{\mathbf{x}}, c_{\mathbf{t}})$ is equal to 1 when $\mathbf{x}$ belongs to the class associated with the cluster centered at $\mathbf{t}$, and 0 if it does not. This error function has twice as many parameters to optimize (both the weights and the center are adjusted), but computational cost of calculations here is linear in the number of vectors $O(n)$, and since only a few iterations are needed this part of learning is quite fast.

If all vectors for which the trained node gives output $+1$ have the same label then this node is a good partial classifier and sequential constructive method described above can be used directly for network construction. However, for some datasets linear projections cannot create pure clusters, as for example in the Concentric Rings case. Creation of a partial classifier may then be done by searching for additional directions by optimization of $Q(\mathbf{t};\mathbf{w})$ function (11) in respect to weights $\mathbf{w}$ and center $\mathbf{t}$ restricted to the subset of vectors that fall into the impure cluster. Resulting direction and center define the next network node according to Eq. 16. If this node is not pure, that is it provides $+1$ output for vectors from more than one class, then more nodes are required. This leads to the creation of a sequence of neurons $\{M_i\}_{i=1}^K$, where the last neuron M_K separates some subset of training vectors without mistakes. Then the following function:

$$\bar{M}(\mathbf{x}) = \begin{cases} +1 & \text{if } \frac{1}{K}\sum_{i=1}^{K} M_i(\mathbf{x}) - \frac{1}{2} > 0 \\ -1 & \text{otherwise} \end{cases} \tag{19}$$

is a partial classifier. In neural network function Eq. 19 is realized by group of neurons M_i placed in the first hidden layer and connected to a threshold node $\bar{M}$ in the second hidden layer with weight equal to $\frac{1}{K}$ and bias $\frac{1}{2}$. This approach has been implemented and the test results are reported below.

QPCNN tests

Table 3 presents comparison of results of the nearest neighbor (1-NN), naive Bayes classifier, support vector machine (SVM) with Gaussian kernel, the *c3sep* network described in this article, and the constructive network based on the QPC index (QPCNN). 9 datasets from the UCI repository [1] have been used in 10-fold crossvalidation to test generalization capabilities of these systems. For the SVM classifier parameters γ and C have always been optimized using an inner 10-fold crossvalidation procedure, and those that produced the lowest error have been used to learn the model on the whole training data.

Most of these datasets are relatively simple and require networks with only a few neurons in the hidden layer. Both the *c3sep* and the QPCNN networks achieve good accuracy, in most cases comparable with 1-NN, Naive Bayes and SVM algorithms. General constructive sequence learning in original formulation applied to QPCNN may lead to overfitting. This effect have ocurred for Glass and Pima-diabetes where average size of created networks is higher than in the case of *c3sep* network, while the average accuracy is lower. To overcome this problem proper stop criterium for growing the network should be considered, e.g. by tracking test error changes estimated on a validation subset.

Table 3 Average classification accuracy for 10 fold crossvalidation test. Results are averaged over 10 trials. For SVM average number of support vectors (#SV) and for neural networks average number of neurons (#N) are reported.

dataset	**1-NN**	**N. Bayes**	**SVM**		**C3SEP**		**QPCNN**	
	acc.	acc.	acc.	# SV	acc.	# N	acc.	# N
Appendicitis	81.3 ± 1.5	85.3 ± 1.0	86.5 ± 0.3	32.1	85.3 ± 1.0	4.2	83.4 ± 1.0	4.0
Flag	50.1 ± 1.1	41.1 ± 1.1	51.1 ± 1.1	315.2	53.6 ± 1.8	26.7	52.9 ± 2.8	10.7
Glass	68.2 ± 1.7	47.4 ± 1.7	66.7 ± 0.9	295.8	61.1 ± 1.3	14.0	57.7 ± 2.3	26.7
Ionosphere	85.2 ± 1.2	82.2 ± 0.2	90.8 ± 1.1	63.9	85.1 ± 1.5	7.9	81.3 ± 1.5	6.3
Iris	95.9 ± 0.5	94.9 ± 0.2	95.5 ± 0.3	43.4	95.7 ± 1.0	5.0	95.3 ± 1.0	3.0
Pima-diabetes	70.5 ± 0.5	75.2 ± 0.5	70.3 ± 1.0	365.3	76.3 ± 0.4	9.1	65.2 ± 0.4	13.1
Promoters	78.5 ± 1.8	85.8 ± 1.3	73.1 ± 1.5	77.2	74.7 ± 5.6	3.7	78.8 ± 2.4	2.6
Sonar	86.8 ± 1.8	67.8 ± 1.2	84.2 ± 1.1	109.7	77.9 ± 2.4	8.5	80.2 ± 2.4	5.1
Wine	95.1 ± 0.8	98.1 ± 0.3	95.1 ± 0.2	63.3	97.1 ± 0.8	4.0	97.4 ± 0.8	3.0

4 Discussion and conclusions

The big challenge facing computational intelligence is to discover correct bias for a given data, creating a simple but accurate models [4]. Many datasets, such as those arising from the natural language processing and problems in bioinformatics, may have an inherent complex logics that we are unable to decipher. This challange has not yet been met by the existing systems and may require a taylor-made methods for a given data that may be created by meta-learning [6, 4]. Neural networks and kernel classifiers are universal approximators and thus they may learn any problem creating a highly complex solution. However, this leads to a poor generalization, because the correct underlying model that represents data cannot be discovered. From Fig. 5 it is evident that an optimal model should use transformations that discover important features, followed in the reduced space by a specific approach, depending on the character of a given data.

Each learning procedure is based on some guiding principles. Minimization of error rarely leads to the discovery of the simplest data models and thus cannot be the only basis for optimal learning systems. Linear separability is also not the best goal for learning. In many cases k-separable solutions are much simpler to achieve, leaving non-separable clusters that are easily handled. They may be treated as strongly regularized (all prototypes on a single line) nearest prototype models. The QPC index provides one way to find k-separable projections. It allows to solve problems that go well beyond capabilities of standard neural networks, such as the classification of Boolean functions in high-dimensional cases. It also enables visualization of data in one or more dimensions, allowing for estimation of reliability of classification for individual cases. It will also be useful for dimensionality reduction and feature selection.

The *c3sep* and QPCNN networks presented in this article are also designed to deal with complex data using a very simple model. The *c3sep* approach tries to find a simplification of the k-separable projection, with each node designed to discriminate a single large cluster. This is done using the error function with additional penalty and reward terms, showing many advantages when dealing with complex logical problems. This network is able to discover simple models for difficult Boolean functions and works also well for real benchmark problems.

Many other variants of the constructive networks based on the guiding principles that may be implemented using projection pursuit indices are possible. The class of PP networks is quite broad. One can implement many transformations in the hidden layer, explicitly creating hidden representations that are used as new inputs for further network layers, or used for initialization of standard networks. Brains are capable of deep learning, with many specific transformations that lead from simple contour detection to final invariant object recognition. Studying lienar and non-linear projection pursuit networks will be most fruitful in combination with the meta-learning techniques, searching for the simplest data models in the low-dimensional spaces after initial PP transformation. This approach should bring us a bit closer to the powerful methods required for deep learning and for discovering hidden knowledge in complex data.

References

1. Asuncion, A., Newman, D.: UCI repository of machine learning databases (2007). URL http://www.ics.uci.edu/∼mlearn/MLRepository.html
2. Bengio, Y., Delalleau, O., Roux, N.L.: The curse of dimensionality for local kernel machines. Tech. Rep. Technical Report 1258, Dṕartement d'informatique et recherche opérationnelle, Université de Montréal (2005)
3. Duch, W.: *k*-separability. Lecture Notes in Computer Science **4131**, 188–197 (2006)
4. Duch, W.: Towards comprehensive foundations of computational intelligence. In: W. Duch, J. Mandziuk (eds.) Challenges for Computational Intelligence, vol. 63, pp. 261–316. Springer (2007)
5. 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**, 277–306 (2001)
6. Duch, W., Grudziński, K.: Meta-learning via search combined with parameter optimization. In: L. Rutkowski, J. Kacprzyk (eds.) Advances in Soft Computing, pp. 13–22. Physica Verlag, Springer, New York (2002)
7. Duch, W., Jankowski, N.: Survey of neural transfer functions. Neural Computing Surveys **2**, 163–213 (1999)
8. Duch, W., Jankowski, N.: Taxonomy of neural transfer functions. In: International Joint Conference on Neural Networks, vol. III, pp. 477–484. IEEE Press, Como, Italy (2000)
9. Duch, W., Jankowski, N.: Transfer functions: hidden possibilities for better neural networks. In: 9th European Symposium on Artificial Neural Networks, pp. 81–94. De-facto publications, Brusells, Belgium (2001)
10. Friedman, J.: Exploratory projection pursuit. Journal of the American Statistical Association **82**, 249–266 (1987)
11. Friedman, J.H., Tukey, J.W.: A projection pursuit algorithm for exploratory data analysis. IEEE Trans. Comput. **23**(9), 881–890 (1974). DOI http://dx.doi.org/10.1109/T-C.1974.224051
12. Grochowski, M., Duch, W.: A Comparison of Methods for Learning of Highly Non-Separable Problems. Lecture Notes in Computer Science **5097**, 566–577 (2008)

13. Grochowski, M., Jankowski, N.: Comparison of instance selection algorithms. ii. results and comments. Lecture Notes in Computer Science **3070**, 580–585 (2004)
14. Haykin, S.: Neural Networks - A Comprehensive Foundation. Maxwell MacMillian Int., New York (1994)
15. Huber, P.J.: Projection pursuit. Annals of Statistics **13**, 435–475 (1985). URL http://www.stat.rutgers.edu/ rebecka/Stat687/huber.pdf
16. Iyoda, E., Nobuhara, H., Hirota, K.: A solution for the n-bit parity problem using a single translated multiplicative neuron. Neural Processing Letters **18**(3), 233–238 (2003)
17. Jankowski, N., Grochowski, M.: Comparison of instance selection algorithms. i. algorithms survey. Lecture Notes in Computer Science **3070**, 598–603 (2004)
18. Jones, C., Sibson, R.: What is projection pursuit. Journal of the Royal Statistical Society A **150**, 1–36 (1987)
19. Kohonen, T.: Self-organizing maps. Springer-Verlag, Heidelberg Berlin (1995)
20. Kordos, M., Duch, W.: Variable Step Search MLP Training Method. International Journal of Information Technology and Intelligent Computing **1**, 45–56 (2006)
21. Liu, D., Hohil, M., Smith, S.: N-bit parity neural networks: new solutions based on linear programming. Neurocomputing **48**, 477–488 (2002)
22. Maass, W., Markram, H.: Theory of the computational function of microcircuit dynamics. In: S. Grillner, A.M. Graybiel (eds.) Microcircuits. The Interface between Neurons and Global Brain Function, pp. 371–392. MIT Press (2006)
23. Marchand, M., Golea, M.: On learning simple neural concepts: from halfspace intersections to neural decision lists. Network: Computation in Neural Systems **4**, 67–85 (1993)
24. Mascioli, F.M.F., Martinelli, G.: A constructive algorithm for binary neural networks: The oil-spot algorithm. IEEE Transactions on Neural Networks **6**(3), 794–797 (1995)
25. Muselli, M.: On sequential construction of binary neural networks. IEEE Transactions on Neural Networks **6**(3), 678–690 (1995)
26. Muselli, M.: Sequential constructive techniques. In: C. Leondes (ed.) Optimization Techniques, vol. 2 of Neural Network Systems, Techniques and Applications, pp. 81–144. Academic Press, San Diego, CA (1998)
27. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes: The Art of Scientific Computing. Cambridge University Press (2007)
28. Sahami, M.: Learning non-linearly separable boolean functions with linear threshold unit trees and madaline-style networks. In: National Conference on Artificial Intelligence, pp. 335–341 (1993)
29. Shakhnarovish, G., Darrell, T., Eds., P.I.: Nearest-Neighbor Methods in Learning and Vision. MIT Press (2005)
30. Stork, D., Allen, J.: How to solve the n-bit parity problem with two hidden units. Neural Networks **5**, 923–926 (1992)
31. Wilamowski, B., Hunter, D.: Solving parity-n problems with feedforward neural network. In: Proc. of the Int. Joint Conf. on Neural Networks (IJCNN'03), vol. I, pp. 2546–2551. IEEE Computer Society Press, Los Alamitos, CA, Portland, Oregon (2003)
32. Young, S., Downs, T.: Improvements and extensions to the constructive algorithm carve. Lecture Notes in Computer Science **1112**, 513–518 (1996)
33. Zollner, R., Schmitz, H.J., Wünsch, F., Krey, U.: Fast generating algorithm for a general three-layer perceptron. Neural Networks **5**(5), 771–777 (1992)