Lecture Notes in Computer Science, Vol. 5097, 566-577, 2008.

A Comparison of Methods for Learning of Highly Non-Separable Problems

Marek Grochowski and Włodzislaw Duch

Department of Informatics, Nicolaus Copernicus University, Toruń, Poland,
Email: grochu@is.umk.pl, Google: W. Duch

Abstract. Learning in cases that are almost linearly separable is easy, but for highly non-separable problems all standard machine learning methods fail. Many strategies to build adaptive systems are based on the "divide-and-conquer" principle. Constructive neural network architectures with novel training methods allow to overcome some drawbacks of standard backpropagation MLP networks. They are able to handle complex multidimensional problems in reasonable time, creating models with small number of neurons. In this paper a comparison of our new constructive *c3sep* algorithm based on k-separability idea with several sequential constructive learning methods is reported. Tests have been performed on parity function, 3 artificial Monks problems, and a few benchmark problems. Simple and accurate solutions have been discovered using *c3sep* algorithm even in highly non-separable cases.

1 Introduction

Multi-layer neural networks, basis set expansion networks, kernel methods and other computational intelligence methods are great tools that enable learning from data [1]. Recently the class of non-separable problems has been characterized more precisely using k-separability index [2, 3] that measures the minimum number of intervals needed to separate pure clusters of data in a single projection. This index divides problems into different classes of complexity. For $k = 2$ problems are separable (a single hyperplane separates data from two classes), but convergence and accuracy of standard methods decreases quickly with growing k-separability index. Parity function for n bit binary strings requires $k = n + 1$ clusters and thus is highly non-separable. Each vector is surrounded by the vectors from the opposite class, and therefore methods based on similarity (including kernel methods) do not generalize and fail, while methods based on discriminating hyperplanes may in principle solve such problems, but require $O(n^2)$ parameters and do not converge easily. Universal approximation does not entail good generalization.

Many statistical and machine learning methods thus fail on problems with inherent complex logic. Training feedforward networks with backpropagation-type algorithms requires specification of correct architecture for a given task. Highly non-separable problems even for relatively modest number of dimensions lead to complex network architectures, and learning in such cases suffers from

high computational costs and difficulties with convergences. Even for a simple XOR problem convergence fails fairly often. Constructive neural techniques allow to overcome some of these drawbacks. Furthermore, simplest models of the data should be thought to avoid overfitting and ensure good generalization. Often quite complex data may be described using a few simple rules [4], therefore using an appropriate constructive strategy should create network with small number of neurons and clear interpretation.

Several sequential constructive learning methods have been proposed [5–11], based on computational geometry ideas. Most of them work only for binary problems and therefore pre-processing using Gray coding is used. However, it is interesting to see how well these algorithms will work on Boolean functions and real benchmark problems, and how do they compare to our *c3sep* algorithm. Next section gives short description of most promising of these methods and the third section contains comparison made on artificial as well as real benchmark problems.

2 Algorithms

All algorithms reported below may be viewed as a realization of general sequential constructive method described by Muselli in [5]. In each step of these algorithms searching for best separation between maximum possible number of vectors of the same class from the rest of the training samples is performed. Consider a separation with hyperplanes realized by threshold neuron:

$$\varphi(\boldsymbol{x}) = \begin{cases} +1 \text{ if } \quad \boldsymbol{w}\boldsymbol{x} - w_0 \geq 0 \\ -1 \text{ otherwise} \end{cases} \tag{1}$$

Let's denote by $\mathcal{Q}^+$ a set of all vectors from a single selected class, by $\mathcal{Q}^-$ the remaining training samples, and by $\mathcal{R}$ a set of training vectors for which the neuron performing separation gives an output equal to $+1$. Then each step of this algorithm could be seen as searching for a subset $\mathcal{R} \in \mathcal{Q}^+$ with maximum number of elements. After each step vectors from the set $\mathcal{R}$ are removed from the training dataset, and if some unclassified patterns still remain the search procedure is repeated. 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), where each separating hyperplane corresponds to a hidden neuron in the final neural network. Weights in the output layer do not take part in the learning phase and their values can be determined from a simple algebraic equation proposed in [5]. The main difference between learning algorithms described here lies in the search method employed for creating the best hidden neuron. Most of those algorithms were created to handle datasets with binary attributes. Some of them (Irregular Partitioning, Carve, Target Switch) have extended versions for the general case of real valued features. Two methods (Sequential Window and *c3sep*) employ

neurons with window-like activation function

$$\tilde{M}_i(\boldsymbol{x}; \boldsymbol{w}, a, b) = \begin{cases} 1 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} \in [a, b] \\ 0 \text{ if } \boldsymbol{w} \cdot \boldsymbol{x} \notin [a, b] \end{cases} \tag{2}$$

while the rest of them use threshold neurons.

2.1 Irregular Partitioning Algorithm

This algorithm [10] starts with an empty set $\mathcal{R}$ and in each iteration one vector from $\mathcal{Q}^+$ is moved to $\mathcal{R}$. A new pattern is kept in $\mathcal{R}$ only if the $\mathcal{R}$ and $\mathcal{Q}^-$ sets are linearly separable. Quality of obtained results and computational cost of this algorithm depends on the type of methods employed to verify existence of the separating hyperplane. Although simple perceptron learning may be used Muselli proposed to use the Thermal Perceptron Learning Rule (TPLR) [5], because it has high convergence speed. In this paper linear programming techniques are used to find an optimal solution.

2.2 Carve Algorithm

The Carve algorithm considers a convex hull generated by the elements of $\mathcal{Q}^-$. Searching for the best linear separation of points from class $\mathcal{Q}^+$ is done here by a proper traversing and rotating of hyperplanes that pass through boundary points of the convex hull. Using the general dimension gift wrapping method a near optimal solution may be found in polynomial time [6].

2.3 Target Switch Algorithm

In every step of the Target Switch Algorithm, proposed by Zollner *et al.* [11] and extended for real valued features by Campbell and Vicente [8] search for the threshold neuron that performs best separation between the set of vectors from a given class $\mathcal{R} \in \mathcal{Q}^+$ and the rest of samples $\mathcal{Q}^-$ is made using any algorithm that searches for linear separability (e.g. perceptron learning, TPLR or some version of linear discrimination method). If the current solution misclassifies some vectors from the $\mathcal{Q}^-$ set the vector from $\mathcal{Q}^+$ nearest to the misclassified vector from $\mathcal{Q}^-$, with the largest distance to the current hyperplane is moved to $\mathcal{Q}^-$, and the learning process is repeated. This procedure leads in a finite number of iterations to a desired solution where on one side of the hyperplane only vectors from a single class are left.

2.4 Oil Spot Algorithm

Data with the binary-valued features may be represented as n-dimensional hypercube. There are two conditions that must be satisfied to separate sets $\mathcal{R}$ and $\mathcal{Q}^-$ with a hyperplane: (1) vectors in R are nodes of connected subgraph, and (2) two parallel edges connecting vectors from $\mathcal{R}$ and $\mathcal{Q}^-$ are always with the same orientation. This concept is used by Oil Spot algorithm to search for optimal separation of binary data with a hyperplane [9].

2.5 Sequential Window Learning

This method uses neurons with window-like transfer functions of the form

$$\varphi(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad |\boldsymbol{w}\boldsymbol{x} - w_0| \leq \delta \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

For this type of neurons fast and efficient learning algorithm for binary valued data based on solution of a system of algebraic equations has been proposed by Muselli [7]. This algorithm starts with two patterns from $\mathcal{Q}^+$ in $\mathcal{R}$ and incrementally adds new vectors from $\mathcal{Q}^+$, maximizing the number of correctly classified vectors by the window-like neuron.

2.6 *c3sep* - Constructive 3-separability Learning

The *c3sep* algorithm [3] uses traditional error minimization algorithm to train neurons with soft-windowed transfer functions, e.g. bicentral functions [12]:

$$\tilde{\varphi}(\boldsymbol{x}) = \sigma(\beta(\boldsymbol{w}\boldsymbol{x} - w_0 - \delta))(1 - \sigma(\beta(\boldsymbol{w}\boldsymbol{x} - w_0 + \delta)) \tag{4}$$

Sharp decision boundaries, like those in Eq. (2), are obtained by using a large value of the slope β at the end of the learning procedure. Function (4) separates a cluster of vectors projected on a line defined by $\boldsymbol{w}$ with boundaries defined by w_0 and δ parameters. This transformation splits input space with two parallel hyperplanes into 3 disjoint subsets. A large and pure cluster $\mathcal{R}$ of vectors from one class is generated by minimization of the following error function:

$$E(\boldsymbol{x}) = \frac{1}{2}\sum_{\boldsymbol{x}}(y(\boldsymbol{x}) - c(\boldsymbol{x}))^2 + \lambda_1 \sum_{\boldsymbol{x}}(1 - c(\boldsymbol{x}))y(\boldsymbol{x}) - \lambda_2 \sum_{\boldsymbol{x}} c(\boldsymbol{x})y(\boldsymbol{x}) \tag{5}$$

where $c(\boldsymbol{x}) = \{0, 1\}$ denotes the label of a vector $\boldsymbol{x}$, and $y(\boldsymbol{x}) = \sum \tilde{\varphi}_i(\boldsymbol{x})$ is the actual output of a network. All weights in output layer are fixed to 1. First term is the standard mean square error (MSE) measure, second term with λ_1 (penalty factor) increases the total error for vectors x_i from the $c(x_i) = 0$ class that fall into cluster of vectors from the opposite class 1 (it is the penalty for "contaminated" clusters), and the 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 the reward for large clusters) [3]. To speed up training procedure, after each learning phase, samples that were correctly handled by previous nodes may be removed from the training data. However, larger clusters (and thus better generalization) may be obtained if these vectors are not removed and different neurons forming clusters that share some vectors are created. The *c3sep* algorithm defines a simple feedforward network with window-line neurons in the hidden layer, and every node trying to find best 3-separable solution in the input space (projection on the line that create large cluster of vectors from one class between two clusters of vectors from the opposite class). This is the simplest extension of linear separability; other goals of learning have been discussed in [3, 13].

3 Results

Results for three types of problems are reported here: Boolean functions offering systematically increasing complexity, artificial symbolic benchmark problems, and real benchmark problems.

3.1 Boolean Functions

First all algorithms described above have been applied to several artificial Boolean functions. In this test no generalization is required since only the complexity and computational cost of models discovered are compared, estimating the capacity of different algorithms to discover simplest correct solutions. Fig. 1 presents results of learning of constructive methods applied to the parity problems from 2 bits (equivalent to the XOR problem) to 10 bits, and Fig. 2 shows the results of learning randomly selected Boolean functions with $P(C(\boldsymbol{x}) = 1) = 0.5$ (the same function is used to train all algorithms). The most likely random function for n bits has k-separability index around $n/2$. 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. It may also be solved by one neuron with n thresholds [2]. All results are averaged over 10 trials.

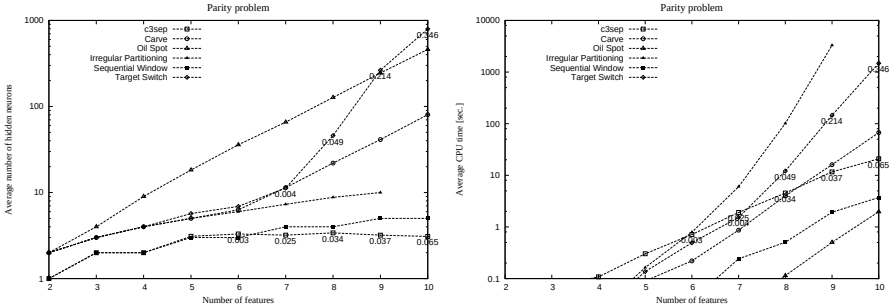

Fig. 1. Number of hidden units created and time consumed during learning of parity problems. Each result is averaged over 10 trials.

Most algorithms are able to learn all parity patterns without mistake. The *c3sep* network, using a stochastic algorithm made in some runs small errors, although with repeating training few times perfect solution is easily found. The target switch algorithm for problems with dimension larger then $n = 7$ generate solution with rapidly growing training errors. Values of the error are placed in corresponding points of Fig. 1 and Fig. 2. Sequential window learning and irregular partitioning algorithms were able do obtain optimal solution for all dimensions.

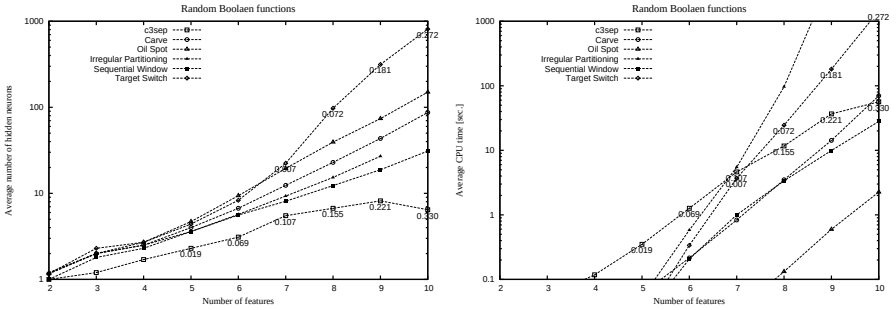

Fig. 2. Number of hidden units created and time consumed during learning of random Boolean functions. Each result is averaged over 10 trials.

Learning of random Boolean function is much more difficult and upper bound for the number of neurons needed for solving of this kind of functions is not known. Learning of irregular partitioning with linear programming for problems with large dimension is time consuming. Oil spot algorithm is the fastest algorithm but leads to networks with high complexity. Sequential window learning gave solutions with a small number of neurons and low computational cost. The Carve algorithm can also handle all patterns in a reasonable time and with a small number of neurons. The *c3sep* network was able to create smallest architectures but the average times of computations are longer than needed by most other algorithms. This network provides near optimal solution, as not all patterns were correctly classified; it avoids small clusters that may reduce classification error but are not likely to generalize well in crossvalidation tests. Algorithms capable of exact learning of every example by creating separate node for single vectors are rarely useful.

4 Benchmark problems

Monk's Problems Monk's problems are three symbolic classification problems designed to test machine learning algorithms [14]. 6 symbolic attributes may be combined in 432 possible ways, and different logical rules are used to divide these cases into two groups, one called "a monk", and the other some other object. The Monk 3 problem is intentionally corrupted by noise, therefore an optimal accuracy is 94%. The other two problems have perfect logical solutions.

Sequential learning algorithm and the Oil Spot algorithm require binary valued features therefore proper transformation of the symbolic data must be applied first. Resulting binary dataset contains 15 inputs where a separate binary feature is associated with presence of each symbolic value. For binary valued features Hamming clustering [15] can be applied to reduce the size of training set and reduce the time of computations. This algorithm iteratively clusters to-

gether all binary strings from the same class that are close in the Hamming distance.

All algorithms have been trained on the 3 Monk's problems 30 times, and averaged results for generalization error, the size of created network and computational costs of training are reported in Tables 4, 4 and 4, respectively. Sequential window learning algorithm and *c3sep* network were able to achieve great accuracy with very small number of neurons. With binary features only a single neuron is sufficient to solve the problem in each case (only for Monk1 sometimes 2 neurons are created), showing that in the binary representation all these problems are 3-separable. For Monk3 problem it is the only algorithm that achieves optimal accuracy (using binary features). Sequential Window algorithm produces also quite simple networks and finds very good solution with Hamming clustering, staying behind *c3sep* in both accuracy and complexity only for the Monk3 problem. The use of the Hamming clustering in most cases increased generalization powers and reduced the number of neurons in the hidden layer. CPU times for all algorithms are comparable.

An example of a projection on the first direction found by the *c3sep* algorithm for the Monk1 problem shows (Fig. 3) a large cluster in the center that can be separated using a window function. If the direction of the projection is binarized logical rules for classification may be extracted.

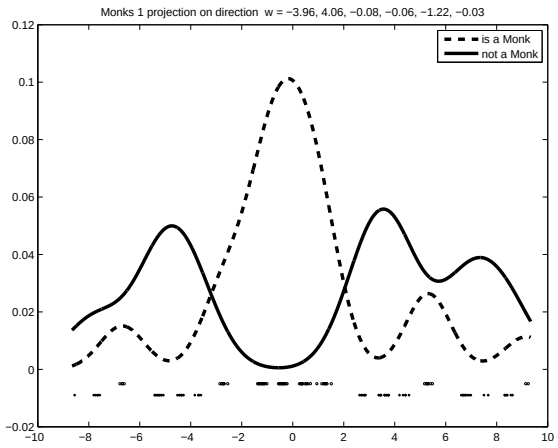

Fig. 3. Probability density for two classes projected on first direction found for the Monk1 problem.

Benchmark Problems with Crossvalidation Tests Boolean problems are rather distinct from typical learning problems. Several tests have been performed on a few benchmark datasets from the UCI repository [16]. The Iris dataset is

Table 1. Accuracy of neural networks for Monk's problems. Each result is averaged over 30 trials.

	Monk's 1	**Monk's 2**	**Monk's 3**
Carve	93.93 ± 2.63	82.42 ± 2.10	88.50 ± 2.47
Irregular Partitioning	87.93 ± 4.19	83.94 ± 1.65	87.99 ± 2.06
Target Switch	93.46 ± 2.57	65.80 ± 2.64	80.38 ± 1.67
c3sep	99.69 ± 1.16	79.78 ± 1.32	87.84 ± 2.36
binary features			
Carve	86.94 ± 3.79	87.14 ± 2.89	85.46 ± 3.09
Oil Spot	86.00 ± 1.63	84.32 ± 0.99	71.84 ± 1.16
Irregular Partitioning	100.00 ± 0.00	98.15 ± 0.00	91.44 ± 1.58
Sequential Window	95.18 ± 5.78	100.00 ± 0.00	84.68 ± 5.82
Target Switch	94.84 ± 3.18	91.23 ± 5.29	90.73 ± 1.97
c3sep	92.49 ± 9.12	100.00 ± 0.00	94.41 ± 0.31
with Hamming clustering			
Carve	100.00 ± 0.00	97.50 ± 0.57	90.29 ± 2.85
Irregular Partitioning	83.84 ± 3.74	99.92 ± 0.46	92.58 ± 1.33
Oil Spot	100.00 ± 0.00	95.07 ± 0.52	90.22 ± 1.15
Sequential Window	100.00 ± 0.00	100.00 ± 0.00	91.72 ± 1.66
Target Switch	100.00 ± 0.00	98.09 ± 0.16	90.62 ± 1.00
c3sep	99.86 ± 0.76	99.38 ± 1.03	91.40 ± 2.54

Table 2. Number of hidden neurons created for Monk's problems. Each result is averaged over 30 trials.

	Monk's 1	**Monk's 2**	**Monk's 3**
Carve	3.30 ± 0.70	10.83 ± 0.99	4.90 ± 0.61
Irregular Partitioning	4.63 ± 0.76	8.77 ± 0.68	4.20 ± 0.41
Target Switch	10.67 ± 2.59	75.73 ± 4.83	12.33 ± 1.83
c3sep	2.43 ± 0.63	2.90 ± 1.94	3.87 ± 0.51
binary features			
Carve	5.30 ± 0.60	8.17 ± 0.99	5.83 ± 0.75
Oil Spot	18.67 ± 2.34	34.37 ± 1.77	31.53 ± 2.29
Irregular Partitioning	3.60 ± 0.62	2.03 ± 0.18	2.00 ± 0.00
Sequential Window	2.70 ± 0.70	1.20 ± 0.41	4.47 ± 0.51
Target Switch	3.73 ± 0.91	12.97 ± 5.04	4.03 ± 0.56
c3sep	1.63 ± 0.49	1.00 ± 0.00	1.00 ± 0.00
with Hamming clustering			
Carve	3.00 ± 0.00	5.00 ± 0.00	5.67 ± 0.61
Oil Spot	3.20 ± 0.41	5.53 ± 0.51	9.20 ± 1.06
Irregular Partitioning	3.00 ± 0.00	5.00 ± 0.00	4.37 ± 0.61
Sequential Window	2.00 ± 0.00	1.00 ± 0.00	5.10 ± 0.96
Target Switch	3.00 ± 0.00	5.00 ± 0.00	5.53 ± 0.63
c3sep	2.00 ± 0.00	1.67 ± 0.88	3.23 ± 0.43

Table 3. CPU time consumed during construction of neural networks solving Monk's problems. Each result is averaged over 30 trials.

	Monk's 1	**Monk's 2**	**Monk's 3**
Carve	0.18 ± 0.02	0.63 ± 0.06	0.21 ± 0.02
Irregular Partitioning	2.04 ± 0.16	8.94 ± 1.64	2.00 ± 0.19
Target Switch	0.53 ± 0.15	6.71 ± 0.66	0.53 ± 0.14
c3sep	0.78 ± 0.19	3.54 ± 1.81	2.36 ± 0.35
binary features			
Carve	0.72 ± 0.09	1.41 ± 0.22	0.69 ± 0.07
Oil Spot	0.18 ± 0.02	0.39 ± 0.06	0.08 ± 0.01
Irregular Partitioning	3.25 ± 0.42	6.32 ± 1.53	2.39 ± 0.35
Sequental Window	1.06 ± 0.34	0.25 ± 0.27	1.72 ± 0.22
Target Switch	0.21 ± 0.08	3.97 ± 1.46	0.06 ± 0.01
c3sep	1.19 ± 0.35	0.73 ± 0.13	1.33 ± 0.12
with Hamming clustering			
Carve	0.01 ± 0.00	0.11 ± 0.01	0.14 ± 0.02
Oil Spot	0.01 ± 0.00	0.02 ± 0.00	0.03 ± 0.00
Irregular Partitioning	0.01 ± 0.00	0.12 ± 0.01	0.11 ± 0.03
Sequental Window	0.01 ± 0.00	0.01 ± 0.00	0.05 ± 0.02
Target Switch	0.01 ± 0.00	0.05 ± 0.00	0.03 ± 0.01
c3sep	0.15 ± 0.06	0.31 ± 0.15	0.45 ± 0.06

perhaps the most widely used simple problem, with 3 types of Iris flowers described by 4 real valued attributes. A Glass identification problem has 9 real valued features with patterns divided into float-processed and non float-processed piece of glass. United States congressional voting record database, denoted here as Voting0 dataset, contains 12 features that determine if a congressman belongs 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 values: yea, nay or missing.

As in the case of Monk's problems some algorithms require additional transformations of datasets. Real valued features were transformed to binary by employing Gray coding. Resulting Iris and Glass dataset in binary representation have 22 and 79 features, respectively. For the three valued input of Voting dataset the same transformation as in the Monk's problems has been adopted. The corresponding binary data contain 48 features for Voting0 and 45 for Voting1 datasets.

The average accuracy obtained after averaging 30 repetitions of the 3-fold crossvalidation test is reported in Tab. 4. The average number of neurons and average CPU time consumed during computations is reported in Tab. 4 and Tab.4. In all this tests in most cases *c3sep* network gave very good accuracy and low variance with very small number of neurons created in the hidden layer. Most algorithms that can handle real features suffer from data binarization, particularly in the case of Iris and Glass where all features in the original data are real valued.

Table 4. 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				
Carve	71.84 ± 3.46	62.08 ± 4.55	91.79 ± 1.22	86.77 ± 1.43
Oil Spot	75.16 ± 2.86	66.05 ± 2.41	90.93 ± 0.90	86.68 ± 1.47
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
with Hamming clustering				
Carve	79.93 ± 3.16	61.92 ± 3.77	93.40 ± 0.69	86.40 ± 1.10
Oil Spot	80.84 ± 2.46	61.83 ± 3.46	92.86 ± 1.09	85.33 ± 1.41
Irregular Partitioning	79.56 ± 4.45	62.38 ± 4.89	93.05 ± 1.10	85.84 ± 1.27
Sequental Window	77.36 ± 4.71	54.18 ± 3.50	91.40 ± 1.21	82.28 ± 1.82
Target Switch	81.09 ± 3.70	62.54 ± 3.57	93.31 ± 0.79	87.02 ± 1.42
c3sep	78.58 ± 4.74	58.51 ± 3.90	83.48 ± 13.39	76.35 ± 11.62

Table 5. 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				
Carve	8.02 ± 0.52	6.79 ± 0.26	5.56 ± 0.32	8.59 ± 0.46
Oil Spot	27.78 ± 1.41	21.54 ± 1.80	22.76 ± 1.39	37.32 ± 2.32
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
with Hamming clustering				
Carve	7.81 ± 0.61	7.28 ± 0.46	5.07 ± 0.37	8.88 ± 0.50
Oil Spot	10.94 ± 1.13	12.17 ± 1.17	7.70 ± 0.74	15.31 ± 1.38
Irregular Partitioning	6.68 ± 0.73	5.39 ± 0.40	4.00 ± 0.48	6.19 ± 0.43
Sequental Window	9.90 ± 0.68	5.54 ± 0.40	5.46 ± 0.50	7.10 ± 0.61
Target Switch	7.02 ± 0.54	8.10 ± 0.63	4.58 ± 0.47	8.98 ± 0.57
c3sep	5.37 ± 0.79	3.38 ± 0.38	3.29 ± 0.40	4.97 ± 0.36

5 Conclusions

Constructive algorithms that may learn functions with inherently complex logics have been collected and compared on learning of Boolean functions, symbolic

Table 6. Average of CPU time consumed during 30x3 CV test.

	Iris	**Glass**	**Voting0**	**Voting1**
Carve	0.07 ± 0.00	0.40 ± 0.02	1.08 ± 0.04	1.85 ± 0.08
Irregular Partitioning	0.64 ± 0.08	4.86 ± 0.57	122.66 ± 24.86	242.48 ± 32.82
Target Switch	0.37 ± 0.05	7.12 ± 1.46	0.08 ± 0.02	1.86 ± 0.37
c3sep	2.05 ± 0.21	1.27 ± 0.17	3.49 ± 0.31	4.58 ± 0.34
binary features				
Carve	1.53 ± 0.09	4.82 ± 0.21	4.22 ± 0.20	5.42 ± 0.24
Oil Spot	0.06 ± 0.00	0.68 ± 0.06	1.74 ± 0.23	2.20 ± 0.40
Irregular Partitioning	0.39 ± 0.02	1.38 ± 0.04	49.34 ± 7.14	474.56 ± 83.94
Target Switch	0.00 ± 0.00	0.02 ± 0.00	0.15 ± 0.04	2.20 ± 0.45
c3sep	2.61 ± 0.34	4.07 ± 0.51	7.28 ± 0.96	7.98 ± 0.18
with Hamming clustering				
Carve	0.28 ± 0.03	0.49 ± 0.04	0.52 ± 0.04	1.47 ± 0.09
Oil Spot	0.04 ± 0.00	0.12 ± 0.01	0.29 ± 0.03	0.62 ± 0.05
Irregular Partitioning	0.51 ± 0.22	3.54 ± 0.21	0.81 ± 0.40	6.59 ± 1.36
Sequental Window	0.29 ± 0.05	1.11 ± 0.16	0.60 ± 0.13	2.52 ± 0.39
Target Switch	0.06 ± 0.02	0.76 ± 0.13	0.29 ± 0.04	0.98 ± 0.12
c3sep	2.29 ± 6.73	0.77 ± 0.07	0.68 ± 0.11	1.78 ± 0.25

and real benchmark problems. These algorithms are rarely used but certainly worth more detailed study. Sequential Window algorithm works particularly well for binary problems but is not competitive for benchmark problems with real-valued features. Note that on the Boolean problems used here for testing all the off-the shelf methods that are found in data mining packages (decision trees, MLPs, SVMs and nearest neighbor methods) will completely fail. Therefore investigation of methods that can handle such data is very important.

The key element is to use non-local projections $\boldsymbol{w} \cdot \boldsymbol{x}$ that does not lead to separable distribution of data, followed by a window that covers the largest pure cluster that can be surrounded by vectors from other classes. Constructive neural networks are well suited to use this type of projections. The Support Vector Neural Training constructive algorithm [17] may also be used here; during training all vectors that are too far from the current decision border, that is excite the sigmoidal neuron too strongly or too weakly, are removed, so the final training involves only the vectors in some margin around the decision border. Another alternative algorithm may be based on modified oblique decision tree (such as OC1 [18]), with each node defining $\boldsymbol{w} \cdot \boldsymbol{x}$ projection, followed by univariate decision tree separating the data along the projection line. Several improvements of the *c3sep* algorithm are also possible, for example by using better search algorithms, second order convergent methods etc. A significant progress in handling problems that require complex logic should be expected along these lines. More tests on ambitious bioinformatics and text analysis problems using such algorithms are needed to show their real potential.

References

1. Cherkassky, V., Mulier, F.: Learning from data. Adaptive and learning systems for signal processing, communications and control. John Wiley & Sons, Inc., New York (1998)
2. Duch, W.: k-separability. Lecture Notes in Computer Science **4131** (2006) 188–197
3. Grochowski, M., Duch, W.: Learning highly non-separable boolean functions using constructive feedforward neural network. Lecture Notes in Computer Science **4668** (2007) 180–189
4. Duch, W., Setiono, R., Zurada, J.: Computational intelligence methods for understanding of data. Proceedings of the IEEE **92**(5) (2004) 771–805
5. 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
6. Young, S., Downs, T.: Improvements and extensions to the constructive algorithm carve. Lecture Notes in Computer Science **1112** (1996) 513–518
7. Muselli, M.: On sequential construction of binary neural networks. IEEE Transactions on Neural Networks **6**(3) (1995) 678–690
8. Campbell, C., Vicente, C.: The target switch algorithm: a constructive learning procedure for feed-forward neural networks. Neural Computations **7**(6) (1995) 1245–1264
9. Mascioli, F.M.F., Martinelli, G.: A constructive algorithm for binary neural networks: The oil-spot algorithm. IEEE Transactions on Neural Networks **6**(3) (1995) 794–797
10. Marchand, M., Golea, M.: On learning simple neural concepts: from halfspace intersections to neural decision lists. Network: Computation in Neural Systems **4** (1993) 67–85
11. Zollner, R., Schmitz, H.J., Wünsch, F., Krey, U.: Fast generating algorithm for a general three-layer perceptron. Neural Networks **5**(5) (1992) 771–777
12. Duch, W., Jankowski, N.: Survey of neural transfer functions. Neural Computing Surveys **2** (1999) 163–213
13. Duch, W.: Towards comprehensive foundations of computational intelligence. In Duch, W., Mandziuk, J., eds.: Challenges for Computational Intelligence. Volume 63. Springer, Berlin, Heidelberg, New York (2007) 261–316
14. *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)
15. Muselli, M., Liberati, D.: Binary rule generation via hamming clustering. IEEE Transactions on Knowledge and Data Engineering **14** (2002) 1258–1268
16. Merz, C., Murphy, P.: UCI repository of machine learning databases (1998-2004) http://www.ics.uci.edu/∼mlearn/MLRepository.html.
17. Duch, W.: Support vector neural training. Lecture Notes in Computer Science **3697** (2005) 67–72
18. Murthy, S., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. Jurnal of Artificial Intelligence Research **2** (1994) 1–32