K-Separability

Włodzisław Duch

Department of Informatics, Nicolaus Copernicus University, Grudziądzka 5, Toruń, Poland,
and School of Computer Engineering, Nanyang Technological University, Singapore
Google: Duch

Abstract. Neural networks use their hidden layers to transform input data into linearly separable data clusters, with a linear or a perceptron type output layer making the final projection on the line perpendicular to the discriminating hyperplane. For complex data with multimodal distributions this transformation is difficult to learn. Projection on $k \geq 2$ line segments is the simplest extension of linear separability, defining much easier goal for the learning process. The difficulty of learning non-linear data distributions is shifted to separation of line intervals, making the main part of the transformation much simpler. For classification of difficult Boolean problems, such as the parity problem, linear projection combined with k-separability is sufficient.

1 Introduction

Many popular classifiers, including MLPs, RBFs, SVMs, decision trees [1], nearest neighbor and other similarity based methods [2,3], require special approaches (architectures, kernels) or cannot handle at all complex problems, such as those exemplified by the parity problem: given a training set of binary strings $\{b_1, b_2...b_n\}$ determine if the number of bits equal to 1 is odd or even. In principle universal approximators, such as neural networks, are capable of handling such problems, and there is a whole literature on architectures and neural activation functions that enable the solution of parity problem. However, solutions proposed so far are manually designed to solve this particular problem, and thus will not work well for slightly different problems of similar kind.

Dealing with difficult learning problems like parity off-the-shelf algorithms (for example those collected in Weka [4] or Ghostminer [5] packages) in the leave-one-out or crossvalidation tests for more than 3-bit problems give results at the baserate (50%) level. Knowing beforehand that the data represents parity problem allows for setting an appropriate MLP architecture to solve it [6,7,8,9,10,11,12,13,14], but for large n in real situation it will be very difficult to guess how to choose an appropriate model. Learning Boolean functions similar to parity may indeed be a great test for methods that try to evolve neural architecture to solve a given problem, but so far no such systems are in sight. The reason for this failure is rather simple: neural and other classifiers try to achieve linear separability, and non-linear separable data may require a non-trivial transformation that is very difficult to learn. Looking at the image of the training data in the space defined by the activity of the hidden layer neurons [15,16] one may notice that a perfect solution is frequently found in the hidden space – all data falls into separate clusters – but the clusters are non-separable, therefore the perceptron output layer is

S. Kollias et al. (Eds.): ICANN 2006, Part I, LNCS 4131, pp. 188–197, 2006.

unable to provide useful results. Changing the goal of learning from linear separability to other forms of separability should make the learning process much easier.

It would be very useful to break the notion of non-linearly separable problems into well defined classes of problems with increasing difficulty. This is done in the next section, where the notion of k-separability is introduced. In the third section this notion is combined with linear projections and applied to the analysis of Boolean functions. Algorithms based on k-separability for general classification problems are outlined in section four, with the last section containing a final discussion.

2 k-Separability

Adaptive systems, such as feedforward neural networks, SVMs, similarity-based methods and other classifiers, use composition of vector mappings

$$Y(\mathbf{X}) = M^{(m)}(M^{(m-1)}...(M^{(2)}(M^{(1)}(\mathbf{X}))...)) \tag{1}$$

to assign a label Y to the vector $\mathbf{X}$. To be completely general direct dependence of mappings on inputs and previous transformations should be considered, for example $M^{(2)}(M^{(1)}(\mathbf{X}), \mathbf{X})$, but for simplicity this will be omitted, considering only strictly layered mappings. These mappings may include standardization, principal component analysis, kernel projections, general basis function expansions or perceptron transformations. $\mathbf{X}^{(i)} = M^{(i)}(\mathbf{X}^{(i-1)})$ is the result of mapping after i transformations steps. For dichotomic problems considered below $Y = \mathbf{X}^{(m)} = \pm 1$.

If the last transformation $Y = M^{(m)}(\mathbf{X}^{(m-1)})$ is based on a squashed linear transformation, for example a perceptron mapping $Y = \tanh(\sum_i W_i \mathbf{X}_i^{(m-1)})$, then the values of Y are projections of $\mathbf{X}$ on the $[-1, +1]$ interval, and a perfect separation of classes means that for some threshold Y_0 all vectors from the Y_+ class are mapped to one side and from the Y_- class to the other side of the interval. This means that the hyperplane $\mathbf{W}$ defined in the $\mathbf{X}^{(m-1)}$ space divides samples from the two classes, and $\mathbf{W} \cdot \mathbf{X}^{(m-1)}$ is simply a projection on the line $\mathbf{W}$ perpendicular to this hyperplane, squashed to the $[-1, +1]$ interval by the hyperbolic tangent or similar function.

General parity problems can be solved in many ways. The simplest solution [13] is to look at the sign of the $\prod_{i=1}^{n}(x_i - t_i)$, with $t_i \in (0, 1)$, that is to use a product neuron without any hidden neurons. This solution is very specific to the parity problem and it cannot be generalized to other Boolean functions. Many such solutions that work only for parity problem have been devised [6]–[14], but the challenge is to provide more general solutions that work also for problems of similar or higher difficulty. Many MLP training algorithms have already some difficulties to solve the XOR problem. RBF network with Gaussian hidden units cannot solve it unless special tricks are used. Solutions based on local functions require here a large number of nodes and examples to learn, while non-local solutions may be expressed in a compact way and need only a few examples (this has been already noted in [17]). Consider the noisy version of the XOR problem (Fig. 1). RBF network with two Gaussian nodes with the same standard deviation σ and linear output provides the following two transformations:

$$\mathbf{X} \to \mathbf{X}^{(1)} = (\exp(-|\mathbf{X} - \mu_1|/2\sigma, \exp(-|\mathbf{X} - \mu_2|/2\sigma) \tag{2}$$

$$\mathbf{X}^{(1)} \to Y = \mathbf{X}^{(2)} = \mathbf{W} \cdot \mathbf{X}^{(1)} \tag{3}$$

The solution obtained using maximum likelihood approach [1] placed one basis function in the middle of left-corner cluster, and the other close to the center, as shown in Fig. 1. Although the network fails to achieve linear separability of the data, it is clear from Fig. 1 that all new data will be properly assigned to one of the 3 clusters formed in the hidden space; in crossvalidation test 100% correct answers are obtained on this basis (searching for the nearest neighbor in the hidden space), while the linear output from the network achieves only 50% accuracy (base rate). If the target $Y = \mathbf{X}^{(2)} = \pm 1$ is desired the linear output provides a hyperplane (in this case a line) that tries to stay at a distance one from all data points. If a separate linear output for each class is used lines representing both outputs are parallel, with identical weights but shifted on two units, as shown in the right Fig. 1. Projecting $\mathbf{X}^{(1)}$ data on these lines gives one interval with the data from first class surrounded by two intervals with the data from the second class, separating the data into 3 intervals.

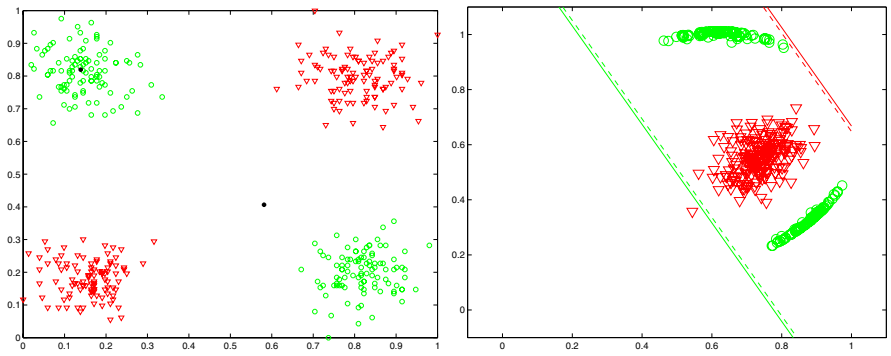

Fig. 1. Noisy XOR problem solved with two Gaussian functions. Left: data distribution and position of two Gaussian functions after training using maximum likelihood principle; right: mapping of the input data to the hidden space shows perfect clusterization, showing lines representing linear weights of the two output units.

In n dimensions a single linear unit $\mathbf{W} \cdot \mathbf{X}$ with all weights $W_i = 1$ easily achieves separation into $n + 1$ groups, with 0, 1, 2 .. n bits equal to 1. This weight vector is the diagonal connecting vertices $[0, 0, \ldots 0]$ and $[1, 1, \ldots 1]$, and $\mathbf{W} \cdot \mathbf{X}$ is the projection on this line. Obviously using a single node with $Y = \cos(\omega \mathbf{W} \cdot \mathbf{X})$ gives for $\omega = \pi$ correct answer to all parity problems, $+1$ for even and -1 for odd number of bits. This is the simplest general solution of the parity problem, using a single node network (this solution has not bee found previously [6]–[14]). The importance of selection of appropriate transfer functions in neural networks is quite evident here (for a taxonomy of transfer functions that may be used in neural networks see [18] and [17]). In the context of Boolean functions periodic projection is useful only for parity and its negation obtained by symmetric transformations of the hypercube with vertices labeled according to their parity. Projections of other Boolean functions may not be periodic but certainly will show several groups of vectors from alternating classes.

There is no particular reason why the target of learning should be linear separability. The last transformation $Y = M^{(m)}(\mathbf{X}^{(m-1)})$ may be designed in any way that will make learning easier. If a projection separating two clusters on a line $Y = \mathbf{W} \cdot \mathbf{X}$ exist the data is linearly separable. If it does not exist a projection forming 3 or more intervals containing clusters from a single class should be sought.

Definition: dataset $\{\mathbf{X}_i\}$ of vectors that belong to two classes is called k-separable if a direction $\mathbf{W}$ exist such that all points $Y_i = \mathbf{W} \cdot \mathbf{X}_i$ are clustered in k intervals, each containing vectors from a single class.

Linearly separable data is called 2-separable, while XOR belongs to the 3-separable category of data distributions, with projection on the $\mathbf{W} = (1, 1)$ line from even, odd, and again even class. This is the simplest extension of separability, replacing the final mapping $M^{(m)}(\cdot)$ by logical rule IF $(Y \in [Y_0, Y_1]$ THEN even ELSE odd, and thus making the non-linearity rather harmless. More sophisticated mappings from one, two or higher number of dimensions may be devised as long as transformation $M^{(m)}(\cdot)$ is easy to set up, providing easier goals for the learning process. The Error Correcting Output Codes (ECOC) [19] tries also to define easier learning targets, but it is still based on linear separability, setting a number of binary targets that define a prototype "class signature" vectors, and comparing the distance from the actual output to these class prototypes. The change of the learning target advocated here is much more powerful.

A dataset that is k-separability may also be $(k + m)$-separable. Strictly speaking the separability index for the data should be taken as the lowest k, but some learning methods may generate solutions with larger number of clusters. For example, if the data is k-separable into clusters with very small number of elements, or if the margin separating the intervals between two such clusters is very small, $(k + 1)$-separability that leads to larger minimum size of small clusters and their margins may be preferred.

Solving a k-separability problem requires finding the direction $\mathbf{W}$ and then setting appropriate $k - 1$ thresholds defining intervals on the projection line.

Conjecture: the complexity of k-separable learning should be much easier then 2-separable learning.

This is rather obvious; transforming the data into k-separability form should be much easier because additional transformations are needed to achieve linear separability, and the number of adaptive parameters may grow significantly. For example, the number of hyperplanes that an MLP network needs for the n-parity problem is of the order of n (see comparison of solutions in [13]), giving altogether $O(n^2)$ parameters, while treating it as a k-separable problem requires only $n + k$ parameters. In general, cases when transformation of decision borders in the original input space $\mathbf{X}$ based on continuous deformations may flatten them linear separability will be sufficient, but if discontinuous transformations are needed, as in the case of learning most Boolean functions, transformations that map data into k-separable form should be easier.

An interesting question is how many Boolean functions belong to the k-separable category. For n-variables there are 2^{2^n} possible functions; only the bounds for the number of separable Boolean functions are known: the number is between $2^{n^2-O(n)}$ and 2^{n^2} [20], a vanishing fraction of all functions. Unfortunately such estimations are not yet known for the k-separable case.

For linearly separable data projections on $\mathbf{W}$ and $-\mathbf{W}$ generate symmetrical solutions (Y_+, Y_-) and (Y_-, Y_+); in case of k-separability additional symmetries and permutations are possible.

3 Boolean Functions

It is instructive to analyze in detail the case of learning Boolean functions with $n = 2$ to $n = 4$ bits with the simplest model based on linear projections. Several interesting questions should be investigated: how many k-separability cases for a given direction $\mathbf{W}$ are obtained; which direction gives the largest separation between projected clusters; how many k-separability cases for each direction $\mathbf{W}$ exist; how many different directions are needed to find all these cases.

The Boolean functions $f(x_1, x_2, \ldots x_n) \in \{-1, +1\}$ are defined on the 2^n vertices of n-dimensional hypercube. Numbering these vertices from 0 to $2^n - 1$, they are easily identified converting decimal numbers to bits, for example vertex 3 corresponds to b-bit string 00..011. There are 2^{2^n} possible Boolean functions, each corresponding to a different distribution of ± 1 values on hypercube vertices. There are always two trivial cases corresponding to functions that are always true and always false, that is 1-separable functions. Each Boolean function may be identified by a number from 0 to $2^n - 1$, or a bit string from 00...0 to 11...1, where the value 0 stands for false or -1, and 1 for true or $+1$. For example, function number 9 has 2^n bits 00...1001, and is true only on vertex number 0 and 3.

Values of Boolean functions may be represented as black (-1) or white $(+1)$ vertices of the hypercube. Learning a Boolean function is equivalent to separation of projections of the black and white vertices of the hypercube. Separation into small number of well separated clusters should lead to a good generalization when some function values are not known. For two binary variables almost all non-canonical directions (not connecting vertices of the square) avoid mapping vertices of different color to exactly the same point (degeneracy) and give 6, 6 and 2 projections 2, 3 and 4-separated, respectively. It is easy to find two directions that together learn all 12 linearly separable functions (for example $\mathbf{W}_{(1/3,-1/2)}$ and orthogonal direction $\mathbf{W}_{(1/3,2/9)}$). These directions and $\mathbf{W}_{(1,1)}$ that learns two 3-separable functions (XOR and its negation) are sufficient to learn all Boolean functions.

3.1 3-D Case

For 3 bits there are 8 vertices in the cube and $2^8 = 256$ possible Boolean functions. Functions $f(x_1, x_2, x_3)$, and their negations $\neg f(x_1, x_2, x_3)$, are related by the sign reversal symmetry or changing color of all vertices, therefore it is sufficient to consider only 128 functions corresponding to all black vertices (1 case), one black vertex (8 cases), two blacks ($\binom{8}{2} = 28$ cases), three blacks ($\binom{8}{3} = 56$ cases), or four blacks ($\binom{8}{4} = 70$ cases, but only half are unique due to the sign reversal symmetry), so together there are 1+8+28+56+35=128 such unique functions.

Projections on coordinate directions $\mathbf{W}_{(001)}, \mathbf{W}_{(010)}, \mathbf{W}_{(100)}$ separates only three functions $f(x_1, x_2, x_3) = x_k, k = 1, 2, 3$ that belong to the 35 cases with 4 black and 4 white vertices. There are 6 projection directions along the diagonals of the cube's

faces: $\mathbf{W}_{(110)}, \mathbf{W}_{(1-10)}, \mathbf{W}_{(101)}, \mathbf{W}_{(10-1)}, \mathbf{W}_{(011)}, \mathbf{W}_{(01-1)}$, and 4 projection directions along the longest diagonals of the cube: $\mathbf{W}_{(111)}, \mathbf{W}_{(11-1)}, \mathbf{W}_{(1-11)}, \mathbf{W}_{(-111)}$. Together 13 canonical directions should be considered, and a "zero direction" to check if there is only one class.

Consider now direction $\mathbf{W}_{(110)}$. Two points, $(0,0,0)$, $(0,0,1)$ are projected to $Y = 0$, 4 points $(1,0,0), (0,1,0), (1,0,1), (0,1,1)$ are at $Y = \sqrt{2}/2$ and two points $(1,1,0)$, $(1,1,1)$ are at $Y = \sqrt{2}$. Any Boolean function that has the same value for the first 6 points and an opposite value for the last 2 point, or vice versa, will be linearly separable. There are 4 such functions. Any function that has the same value at the middle 4 vertices, and an opposite on the remaining 4 will be 3-separable. There are 2 such functions. However, separation and size of the clusters should also be noted. For example, function 27 (00011011) is separated by $\mathbf{W} = [0.75, 1, -0.25]$ into 000 11 0 11 segments with minimum gap of 1/4 and by $\mathbf{W} = [1, 0.25, -0.75]$ into 0 1 000 111 segments with the same minimum gap. The first projection contains only one group with single 0, while the second contains two such groups, one with 0 and one with 1. For Boolean functions with small number of bits generalization is meaningless (there is no evidence to choose a particular function), but for larger number of bits avoiding small clusters should give a better chance to find most probable functions even if some values are missing. For example, for the n-bit parity if some of the values on vertices with $m \in [3, n-2]$ bits 1 are missing projection on $\mathbf{W} = [11 \ldots 11]$ will still provide the best explanation of the data separating it into $n+1$ intervals.

If degeneracy is removed by slightly shifting $0, \pm 1$ weight values of canonical directions (adding 0.01 to the first, 0.02 to the second and 0.03 to the weights $\mathbf{W}$ is sufficient) for an arbitrary projection direction always the same number of 1 to 8-separable functions is found: 2, 14, 42, 70, 70, 42, 14, 2. Thus for a projection on an arbitrary direction most functions are 4 or 5-separable. Searching for the best projection for each function using slightly perturbed canonical directions there are 2 cases of 1-separable functions, and 102, 126 and 26 of 2, 3 and 4-separable functions. For more than half of the 3-bit Boolean functions there is no linear projection that will separate the data. Almost half (126) of all functions may be learned using 3-separability. Because there are 102 linearly separable functions and each projection can recognize only 14 of them at least 8 directions are needed to check whether the function is separable.

3.2 4-D Case

For the 4-bit problem there are 16 hypercube vertices, with Boolean functions corresponding to 16-bit numbers, from 0 to 65535, or 64K functions. Projection on each fixed direction gives symmetric distribution of the number of k-separability functions, with the same number of functions for k and $17-k$ separability. Two functions are 1-separable and two are 16-separable, changing periodically all 16 values from 0 to 1. Linear separation (and 15-separability) is found only for 30 functions, 3-separability for 210, 4 to 8 separability for 910, 2730, 6006, 10010 and 12870 functions respectively. Thus a random initialization of a single perceptron has the highest chance of creating 8 or 9 clusters in the 4-bit data.

Checking how many functions are k-separable requires learning the best direction for a given data. For the 4-bit case searching for the best projection along canonical

directions ($W_i = 0, \pm 1$) that give lowest k-separability index gives 1228, 6836, 19110, 25198, 12014, 1132 and 16 projections with 2-8 clusters. These are not yet the lowest separability indices for this data, as more detailed search allowing fractional values (multiples of 1/3, 1/4, 1/5 and 1/6 in the $[-1, +1]$ range) of the $\mathbf{W}$ direction coefficients shows that the highest k is 5, confirming the suspicion that $k = n + 1$ is the highest separability index. The number of linearly separable functions is 1880, or less than 3% of all functions, with about 22%, 45% and 29% being 3 to 5-separable. About 188 functions were found that seem to be either 4 or 5-separable, but in fact contain projection of at least two hypercube vertices with different labels on the same point. Although the percentage of linearly separated functions rapidly decreases relatively low k-separability indices resolve most of the Boolean functions.

An algorithm that searches for lowest k but also maximizes minimum distance between projections of points with different labels finds projection directions (with minimum separation of 1/6 or more) that require $k = 6$ for these functions and gives significantly larger separations between intervals containing vectors from a single class. With only 30 linarly separable functions per one direction and 1880 separable functions at least 63 different directions should be considered to find out if the function is really linearly separable. Learning all these functions is already a difficult problem.

For 5-bits the number of all Boolean functions grows to 2^{32}, or over 4 billions (4G). Direct search in 5-dimensional space for each of these functions is already prohibitively expensive. It seems quite likely that for n-bit Boolean functions each projection direction will separate the maximum number of functions for $k \approx 2^n/2$, and that learning the best projection for a given function will give the largest number of functions separated into n clusters, with percentage of linearly separable functions going quickly to zero. The number of elements in most cluster quickly grows, therefore with such as simple model it should be possible to learn them correctly even if only a subset of all values is given.

4 Algorithms Based on *k*-Separability

Linear projection combined with k-separability already gives quite powerful learning system, but almost all computational intelligence algorithms may implement in some form k-separability as a target for learning. It is recommended to search first for linearly separable solutions, and then to increase k searching for the simplest solution, selecting the best model using crossvalidation or measures taking into account the size and separation between projected clusters. Distribution of $y(bX; \mathbf{W})$ values allows for calculation of $P(y|Y_{\pm})$ class distributions and posterior probabilities using Bayesian rules. Estimation of probability distributions in one dimension is easy and may be done using Parzen-windows kernel methods.

The main difficulty in formulating a learning procedure is the fact that targets are not fully specified; instead of a single target for Y_+ class two or more labels Y_{+1}, Y_{+2} may be needed. This may actually be of some advantage, allowing for a better interpretation of the results. It is clearly visible in the case of parity problems: each group differs not only by the parity but also by different number of 1's, providing an additional label. Learning should therefore combine unsupervised and supervised components. In the first step random initialization is performed several times, selecting the lowest k cluster

projection. Centers of these clusters $t_i, i = 1 \ldots k$ are the target variables for learning, and each center has a class label $Y(t_i)$. Slightly modified quadratic error function may be used for learning:

$$E(\mathbf{W}, \mathbf{t}) = \frac{1}{2} \sum_{\mathbf{X}} \left(y(\mathbf{X}; \mathbf{W}) - t_j(\mathbf{X}) \right)^2 ; \tag{4}$$
$$j = \arg\min_i \{ ||t_i - y(\mathbf{X}; \mathbf{W})||, Y(t_i) = Y(\mathbf{X}) \}$$

For each input $\mathbf{X}$ that belongs to the class $Y(\mathbf{X})$ the nearest (on the projected line) cluster center from the same class is taken as the learning target. A more complex cost function may be devised that penalize for the number of clusters, for overlapping of clusters, and for impurity of clusters, but this is beyond the scope of this article.

In the two-class case there are always two possibilities: either the first class vectors are projected to the lowest Y values, leading to clusters $Y_+, Y_-, Y_+, \ldots$, or vice versa, $Y_-, Y_+, Y_-, \ldots$. The 3-separable case is particularly simple and often encountered in practice. If vectors from one of the classes represent unusual objects or states (for example hypo and hiper-activity in some medical problems) projections with clusters Y_-, Y_+, Y_- are fairly common. This may be checked quite easily visualizing distribution of activations for a single perceptron (linear neuron is sufficient). Additional transformations (network layers) are needed to reach linear separability, but 3-separability may often be reached using just one node.

For $k = 3$ these projections are in 3 intervals: $[-\infty, a], [a, b], [b, +\infty]$. Taking $t = (a + b)/2$, and denoting $Y_X = Y(\mathbf{X})$ a linear error function suitable for learning is:

$$\begin{aligned} E(a, b, \mathbf{W}) \; = & \sum_{\mathbf{X}} [\mathrm{T}(y \le t) \delta(Y_X, Y_+) \max(0, y - a) + \delta(Y_X, Y_-) \max(0, a - y) \\ & + \mathrm{T}(y > t) \delta(Y_X, Y_+) \max(0, b - y) + \delta(Y_X, Y_-) \max(0, y - b)] \end{aligned} \tag{5}$$

where $\mathrm{T}(y > t)$ is 0 if false and 1 if true. This function admits a trivial $\mathbf{W} = 0$ solution, therefore either a condition $||\mathbf{W}||$ should be introduced, or a distance scale should be fixed by requiring one of the components to be constant. It assumes that Y_+ vectors contribute to errors only outside of the $[a, b]$ interval, with the error growing in a linear way, and that Y_- vectors contribute to error in the linear way only inside this interval. It requires good initialization to map all Y_+ vectors to correct side of t. Using this function for 3-separable Boolean functions with multiple starts to find approximate 3-separability projection quickly learns such functions using a simple gradient method. To avoid threshold functions $\mathrm{T}(y > t)$ may be replaced by a logistic function $\sigma(y - t)$.

3-separable backpropagation learning in purely neural architecture requires a single perceptron for projection plus a combination of two neurons creating a "soft trapezoidal window" type of function $F(Y; a, b) = \sigma(Y + a) - \sigma(Y + b)$ that implements interval $[a, b]$ [21]. These additional neurons (Fig. 2) have fixed weights ($+1$ and -1) and biases a, b, adding only 2 adaptive parameters. An additional parameter determining the slope of the window shoulders may be introduced to scale the Y values. The input layer may of course be replaced by a hidden layer that implements additional mapping.

This network architecture has $n + 2$ parameters and is able to separate a single class bordered by vectors from other classes. For n-dimensional problem with 3-separable

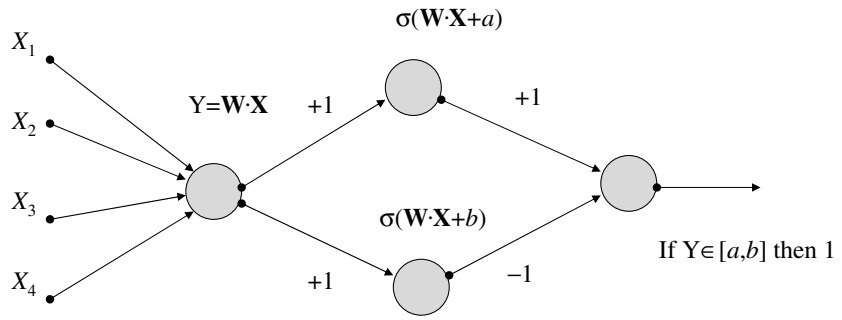

Fig. 2. MLP solution to the 3-separable case of 4-bit Boolean functions

structure standard architecture requires at least two hidden neurons connected to an output neuron with $2(n+1)+3$ parameters. For k-separability case this architecture will simply add one additional neuron for each new interval, with one bias parameter. n-bit parity problems require only n neurons (one linear perceptron and $n-1$ neurons with adaptive biases for intervals), while in the standard approach $O(n^2)$ parameters are needed [13].

5 Discussion and Open Problems

A radically new approach to learning has been proposed, simplifying the process by changing the goal of learning to easier target and handling the remaining nonlinearities with well defined structure. k-separability is a powerful concept that will be very useful for computational learning theory, breaking the space of non-separable functions into subclasses that may be separated into more than two parts. Even the simplest linear realization of k-separability with interval nonlinearities is quite powerful, allowing for efficient learning of difficult Boolean functions. Multiple-threshold perceptrons [22] may implement such intervals, although k-separability learning algorithms require more than multiple-threshold step functions. So far there are no systems that can routinely handle difficult Boolean functions, despite a lot of effort devoted to special Boolean problems, such as the parity problem. Using neural algorithms or special error functions described in the previous chapter almost all $n=4$ Boolean functions have been learned in less than 40 trials (Duch and Adamczak, in preparation).

Redefining the goal of learning may have some biological justification. Neurons in association cortex form strongly connected microcuircuits found in minicolumns, resonating with different frequencies when an incoming signal $X(t)$ appears. A perceptron neuron observing the activity of a minicolumn containing many microcircuits learns to react to signals in an interval around particular frequency. Combination of outputs from selected perceptron neurons is used to discover a category. These outputs may come from resonators of different frequencies, implementing an analogue to the combination of disjoint projections on the $\mathbf{W} \cdot \mathbf{X}$ line.

An interesting concept creates many open problems. How many boolean function each direction k-separates in general case? What minimal k is sufficient for n-bit problems? How will different cost functions perform in practice? What other simple ways

to "disarm" linearites, besides projection on a k-segment line, may be used? These and many other questions will be addressed soon.

Acknowledgement. Support by the Polish Committee for Scientific Research, research grant 2005-2007, is gratefully acknowledged.

References

1. Duda, R.O., Hart, P.E., Stork, D.: Patter Classification. J. Wiley & Sons, New York (2001)
2. Duch, W.: Similarity based methods: a general framework for classification, approximation and association. Control and Cybernetics **29** (2000) 937–968
3. Duch, W., Adamczak, R., Diercksen, G.: Classification, association and pattern completion using neural similarity based methods. Applied Math. & Comp. Science **10** (2000) 101–120
4. Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco (2nd Ed, 2005)
5. Jankowski, N., Grąbczewski, K., Duch, W., Naud, Adamczak, R.: Ghostminer data mining software. Technical report (2000-2005) http://www.fqspl.com.pl/ghostminer/.
6. Stork, D., Allen, J.: How to solve the n-bit parity problem with two hidden units. Neural Networks **5** (1992) 923–926
7. Minor, J.: Parity with two layer feedforward nets. Neural Networks **6** (1993) 705–707
8. Setiono, R.: On the solution of the parity problem by a single hidden layer feedforward neural network. Neurocomputing **16** (1997) 225–235
9. Lavretsky, E.: On the exact solution of the parity-n problem using ordered neural networks. Neural Networks **13** (2000) 643–649
10. Arslanov, M., Ashigaliev, D., Ismail, E.: N-bit parity ordered neural networks. Neurocomputing **48** (2002) 1053–1056
11. Liu, D., Hohil, M., Smith, S.: N-bit parity neural networks: new solutions based on linear programming. Neurocomputing **48** (2002) 477–488
12. Torres-Moreno, J., Aguilar, J., Gordon, M.: The minimum number of errors in the n-parity and its solution with an incremental neural network. Neural Proc. Letters **16** (2002) 201–210
13. 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
14. Wilamowski, B., Hunter, D.: Solving parity-n problems with feedforward neural network. Int. Joint Conf. on Neural Networks (IJCNN'03), Portland, Oregon 2003, Vol I, 2546–2551
15. Duch, W.: Visualization of hidden node activity in neural networks: I. Visualization methods. II. Application to RBF networks. Springer Lecture Notes in AI 3070 (2004) 38–49
16. Duch, W.: Coloring black boxes: visualization of neural network decisions. In: Int. Joint Conf. on Neural Networks, Portland, Oregon. IEEE Press Vol I (2003) 1735–1740
17. Duch, W., Jankowski, N.: Survey of neural transfer functions. Neural Computing Surveys **2** (1999) 163-213
18. Duch, W., Jankowski, N.: Taxonomy of neural transfer functions. In: International Joint Conference on Neural Networks. Como, Italy, IEEE Press Vol III (2000) 477–484
19. Dietterich, T., Bakiri, G.: Solving multiclass learning problems via error-correcting output codes. Journal Of Artificial Intelligence Research **2** (1995) 263–286
20. Zuyev, Y.: Asymptotics of the logarithm of the number of threshold functions of the algebra of logic. Soviet Mathematics Doklady **39** (1989)
21. 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
22. Ngom, A., Stojmenovic I., Zunic, J.: On the Number of Multilinear Partitions and the Computing Capacity of Multiple-Valued Multiple-Threshold Perceptrons, IEEE Transactions on Neural Networks **14** (2003) 469–477