2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

A New Constructive Approach for Creating All Linearly Separable (Threshold) Functions

Leonardo Franco, *Member, IEEE*, José Luis Subirats, Martin Anthony, and José M. Jerez

***Abstract*— A new constructive approach for creating all linearly separable functions is introduced. Balanced and unate functions on N+1 variables are created and then projected to N variables permitting to create straightforward all the linearly separable functions without needing to check for linearly separability. The method is supported by the demonstration of a theorem and numerical simulation results for small number of variables. If the results extrapolates the method may permit to test the linear separability of any function on N variables by checking monotonicity and unateness in N+1 dimension. Furthermore, the generalization complexity of the linear threshold functions is analyzed.**

I. Introduction

Counting the number of linear threshold functions is a complicate and open problem and the number of linearly separable functions is only know up to N=8 [1], [2], [3], [4]. There exist some lower and upper bounds ([5],[6]) but they are not very accurate [7]. Linearly separable (LS) functions, also known as Threshold functions are very important as they are the basics elements of threshold logic circuits from which any other Boolean functions can be constructed. They are also very close related to Neural Networks, in fact if threshold gates are used in the neural networks then threshold circuits are equivalent to them. Standard feed-forward neural networks use sigmoidal activation functions because training is easier with them and also because sigmoidal functions are a bit more powerful in terms of computational capabilities [8], but they are in any case very close in terms of their properties to networks constructed with threshold gates. In the 60's there was a lot of excitement for the construction of VLSI circuits using threshold units instead of Boolean gates, but this excitement vanished as it was technologically difficult and problematic to build a computer using threshold gates. In the 90's with the advance of technology, a new impulse and more attention has been again given to the study of threshold circuits and of linearly separable functions [7], [3], [4], [9], [10].

In this paper we first use symmetry properties of connected graphs to analyze the number of threshold functions as a function of their weight (number of 1's in the output values). It is observed that the number of balanced threshold functions on N+1 variables is the same as the total number of threshold functions in N variables. This result is later enunciated as a theorem and proved. Numerical simulations results are also presented showing that the whole set of threshold Boolean functions can be created straightforward for small dimensions cases by requiring the graphs to be connected, balanced and unate. The method, if the results extrapolate to larger dimensions, would eventually permit to check the linear separability of any Boolean function by testing if function is unate and balanced.

Also using a recently introduced complexity measure [11], [12] related to the generalization ability that can be obtained when Boolean functions are implemented in predictive systems trained by examples (like NNs, SVMs, decision trees, etc.) some properties of the LS functions are also analyzed.

II. The Number of Threshold Boolean Function as a Function of the Weight

A Boolean function function is a map of $\{0,1\}^N \rightarrow \{0,1\}$. For a function of N variables exist 2^N input examples and their corresponding outputs define an specific Boolean function out of the 2^{2^N} existing ones. We will refer in this paper to the weight of a function as the number 1's present in the definition of the function. In particular, a balanced Boolean function is one such that the number of 0's and 1's in the definition of the function are equal. We will also use a representation of graphs that is totally equivalent to define a Boolean function, as the graph connects all outputs of the functions that are equal to 0 (or equivalently to 1).

We first show, for Boolean threshold functions of up to N=4 variables, how to compute the number of functions that exist for a given weight value using a method that relies on symmetry properties. The method is a constructive approach in which for lower dimensions the number of LS functions with weight 0 up to $2^{N-1}$is analyzed. As the weight of the functions to be considered increase, it is only

Leonardo Franco is with the Department of Languages and Computer Science, Campus de Teatinos S/N, University of Málaga, 29071, Málaga, SPAIN (phone: +34-952-133304; fax: +34-952-131397; email: lfranco@lcc.uma.es).

José Luis Subirats is with the Department of Languages and Computer Science, Campus de Teatinos S/N, University of Málaga, 29071, Málaga, SPAIN (phone: +34-952-133304; fax: +34-952-131397; email: josoluco@hotmail.com).

Martin Anthony is with the Department of Mathematics at London School of Economics and Political Science, Houghton Street, London WC2A 2AE, UK, (phone: +44-20-79557623; Fax: +44-20-79556877; email: m.anthony@lse.ac.uk).

José M. Jerez is with the Department of Languages and Computer Science, Campus de Teatinos S/N, University of Málaga, 29071, Málaga, SPAIN (phone: +34-952-132956; fax: +34-952-131397; (email: jja@lcc.uma.es).

0-7803-9490-9/06/$20.00/©2006 IEEE

necessary to consider the extensions of the lower weight cases already considered as if lower weight graphs are not LS their extensions would be whether non LS functions or would be cases that will appear from the enlargement of other LS functions.

Functions up to weight 2^{N-1} are considered as higher cases are totally symmetric to lower ones. The number of Boolean LS functions with weight 2^N is exactly the same as the number with weight zero, as this arise for counting the number of 1's or 0's in the definition of the function and the convention is totally arbitrary. In general the symmetry holds in the sense that the number of cases with weight i and weight 2^{N-i} is the same. A special case is the larger one needed to be considered for any dimension N and is the case of counting functions with weight 2^{N-1}.

A. The case N=1

There are 4 Boolean functions in dimension N=1 for which exists two input examples.

- **Weight 0:** This case is almost trivial. There exist only one such a function, not only for $N = 1$ but in every dimension.
- **Weight 1:** This case is also trivial. The two different inputs can take alternatively the value 1 and thus there are 2 LS functions. For the general case of dimension N there are 2^N LS functions of weight 1.

B. The case N=2

There are 16 Boolean functions in this case in which there are 4 different input examples. As said before the method is constructive and thus only the cases with weight larger than the maximum consider in the previous dimension and between $2^{N-1} = 2$ need to be considered. For N=2, we only need to consider the case of weight 2.

Note that the non LS functions in dimension $N = 2$, the XOR and its negation are not even considered by the method as the two examples having output 1 are not contiguous examples.

- **Weight 2:** There is only one kind of graph that permit to generate the 4 LS functions. The graph is the one that connects two contiguous examples. For the general case in dimension N, there exists 2^N inputs, each one having N nearest neighbors. As pairs are considered, the total number of functions has to be normalized by 2 and thus there are $\frac{2^N N}{2} = N\,2^{N-1}$ LS functions of N variables and weight 2.

C. The case N=3

There are a total of 256 Boolean functions and 8 different input examples in this case. We consider only functions with weight 3 and 4.

- **Weight 3:** With only one kind of graph all functions can be generated, those with all three contiguous inputs equal to 1. There are 24 such a functions for $N = 3$, and one of them is depicted in Fig. 1. For the general case in dimension N the number of LS functions is $2^N \binom{N}{2}$.

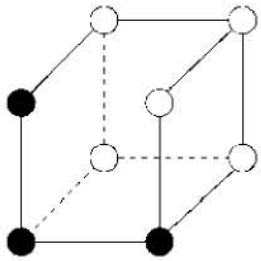

Fig. 1. Graph for N=3 with weight 3. There are 24 similar graphs (or equivalently LS Bolean functions).

- **Weight 4:** There are two kinds of graphs needed to be considered:
  a) Functions that have three example on the 3 different dimensions equal to 1, for which there are 8 cases (equal to the number of corners.
  b) Functions that can be represented by a graph belonging to the face of a cube.

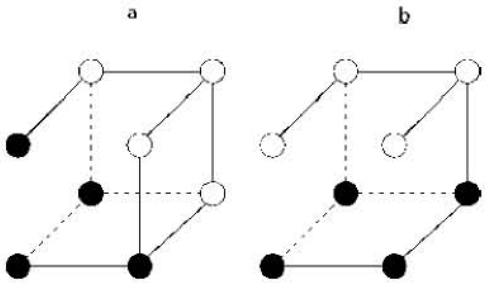

Fig. 2. The two graphs for N=3 with weight 4. There are 24 similar graphs for the one as a) and 86 for the right one.

D. The case N=4

There are 65536 Boolean functions and 16 different input examples in this case. We only need to consider functions with weight $5, 6, 7$ and 8. We will use a way to represent functions in dimension $N = 4$ by joining two 3-D cubes by edges connecting inputs at Hamming distance 1 (neighbors inputs).

- **Weight 5:** LS functions can be generated from two kinds of graphs (See Fig. 3). The total number of LS

functions is 208 for this case.
a) The examples occupies a face of a 3-D cube and also includes a contiguous example. There are 196 such functions.
b) Starting from any one example the other four are in the four different dimensions. 16 LS functions exist.

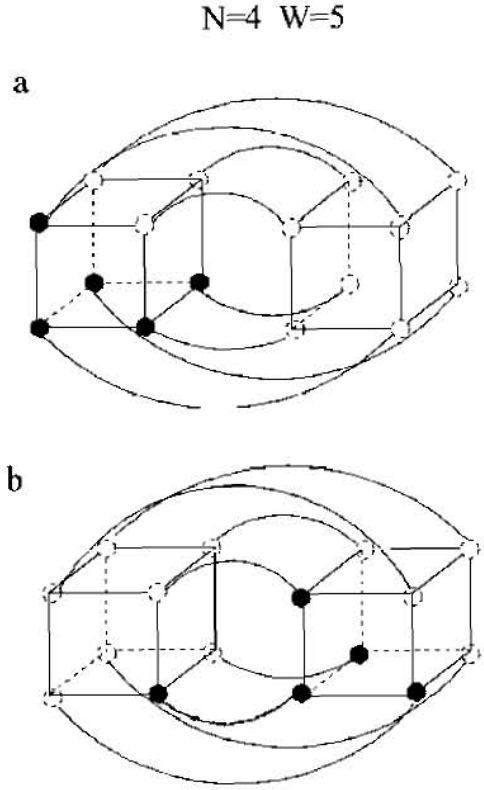

Fig. 3. The two kind of graphs for N=4 with weight 5 with which are possible to generate the 208 LS functions existing in this case. There are 192 similar graphs for the one indicated as a) and 16 for the indicated as b)

- **Weight 6:** Two classes of graphs generate the 192 LS functions existing for this case. (See Fig. 4)
  a) Functions that have all the ones occupying the two contiguous faces of a cube. 72 such functions exists.
  b) Functions that can be represented by a graph where the 1's belong to the face of a cube plus two other ones in orthogonal dimensions. 96 functions constructed.

- **Weight 7:** Only one kind of graph is needed to generate all LS functions in this case, examples occupying a whole 3-D cube except for one corner. There are 64 different graphs.
- **Weight 8:** There are two kinds of graphs that need to be considered to generate the 104 existing LS functions.
  a) Graphs with all 1's occupying a whole 3-D cube, from which 8 different LS functions can be generated.
  b) Functions that can be represented by the edge formed by any two contiguous examples and the three 2-D faces that share this edge. One of this graphs is depicted in Fig. 5. There are 96 such graphs.

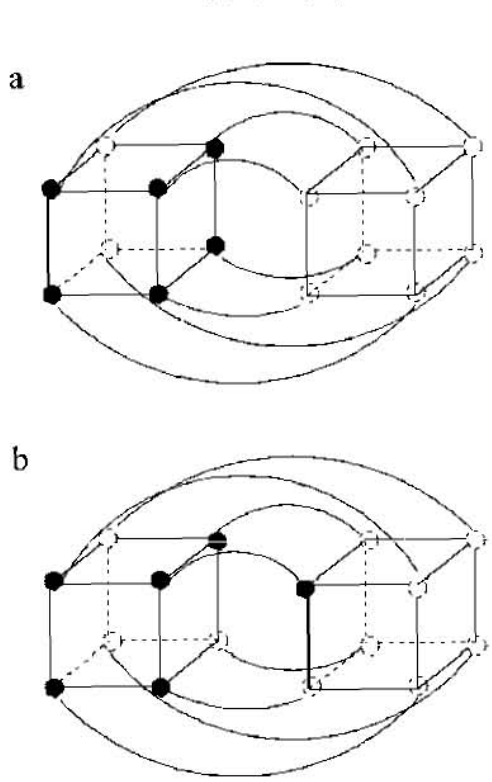

Fig. 4. The two kind of graphs for N=4 with weight 6 with which are possible to generate the 192 LS functions existing in this case. There are 72 similar graphs for the one indicated as a) and 96 for the one indicated as b)

E. The case N=5

There are $2^{2^5} = 4294967295$ Boolean functions in dimension 5 from which only 94572 are LS. To obtain them all we need to consider functions with weight between 9 and 16. The calculation gets too complicated for this case and can be done along the same lines as shown for the case $N = 4$, and thus we decided just to show in table 1, the number of the LS functions obtained for this case.

III. A THEOREM ON THE NUMBER OF LINEARLY SEPARABLE FUNCTIONS

From table 1 is possible to see that for the analyzed cases the number of threshold functions on N variables is the same as the number of threshold functions in N+1 variables with weight 2^N.

Recall that a Boolean function f on N variables is said to be *balanced* if the number of 1's, or *positive examples*, (those $x \in \{0,1\}^N$ for which $f(x) = 1$) is exactly 2^{N-1}, so that the function has an equal number of positive and negative examples.

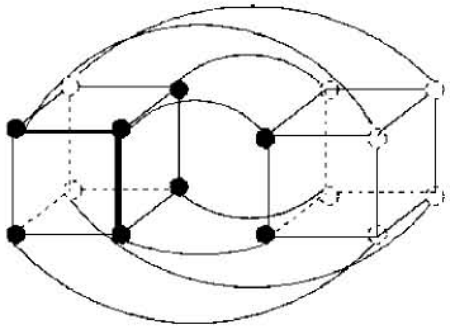

Fig. 5. One of the graph for N=4 with weight 8 with which are possible to generate the 94 LS functions of the 104 existing with weight 8. The common edge for the three faces mentioned in the text is highlighted with a thicker line.

Any linear threshold function $t : \{0,1\}^N \to \{0,1\}$ can be represented by a weight vector $w \in \mathbb{R}^N$ and threshold $\theta \in \mathbb{R}$ such that $t(x) = 1$ if $\langle w, x\rangle > \theta$ and $t(x) = 0$ if $\langle w, x\rangle < \theta$, where $\langle w, x\rangle$ is the usual inner product, $\langle w, x\rangle = w^T x$. (We can, by the discreteness of $\{0,1\}^N$, assume that $\langle w, x\rangle \neq \theta$ for all x.) We write $t \leftarrow [w, \theta]$. (Of course, there are infinitely many possible weight vectors and thresholds that will represent a given t.)

Theorem 3.1: The number of threshold functions on $\{0,1\}^N$ is the same as the number of balanced threshold functions on $\{0,1\}^{N+1}$.

Proof: We construct a bijection from T_N, the set of threshold functions on $\{0,1\}^N$ to BT_{N+1}, the set of balanced threshold functions on $\{0,1\}^{N+1}$. To define this mapping, let us first say that x and y in $\{0,1\}^N$ are *antipodal* if $y = \mathbf{1} - x$, where $\mathbf{1}$ is the all-1 vector. This means that $y_i = 1 - x_i$ for each i. We write $y = \bar{x}$. For $t \in T_N$, define a function $t^* : \{0,1\}^{N-1} \to \{0,1\}$ by:

$$\text{for } x \in \{0,1\}^N, \quad t^*(x0) = t(x), \text{ and } t^*(x1) = 1 - t(x).$$

We first show that t^* is a threshold function. (We note, as an aside, that t^* is also a *self-dual* Boolean function.) We can demonstrate this explicitly by finding a weight vector and threshold that represent t^*. Suppose $t \leftarrow [w, \theta]$. Let $w^* \in \mathbb{R}^{N+1}$ be $w^* = (w, \alpha)$ where $\alpha = 2\theta - \sum_{i=1}^{N} w_i$. (Here, (w, α) means the vector with first N entries w and last entry α.) Now, for any $x = x_1 \ldots x_N \in \{0,1\}^N$, we have

$$t^*(x0) = 1 \iff t(x) = 1 \iff \langle w, x\rangle > \theta \iff \langle w^*, x0\rangle > \theta,$$

and

$$\begin{aligned} t^*(x1) = 1 &\iff t(\bar{x}) = 0 \\ &\iff \langle w, \mathbf{1} - x\rangle < \theta \\ &\iff \sum_{i=1}^{N} w_i - \langle w, x\rangle < \theta \\ &\iff \langle w, x\rangle - \sum_{i=1}^{N} w_i + 2\theta > \theta \\ &\iff \langle w^*, x1\rangle > \theta. \end{aligned}$$

So we see that $t^* \leftarrow [w^*, \theta]$ is indeed a threshold function.

The definition of t^* shows that the mapping $t \mapsto t^*$ is injective; for, if $t_1^* = t_2^*$ then for all $x \in \{0,1\}^N$ we have

$$t_1(x) = t_1^*(x0) = t_2^*(x0) = t_2(x).$$

TABLE I

THE NUMBER OF LINEAR SEPARABLE (OR THRESHOLD) BOOLEAN FUNCTIONS AS A FUNCTION OF THE WEIGHT OF THE FUNCTIONS FOR DIMENSIONS $N = 1, 2, 3, 4$ & 5. FOR EACH COLUMN WITH A GIVEN VALUE OF N THE NUMBER OF FUNCTIONS WITH WEIGHT 2^{N-1} IS HIGHLIGHTED IN BOLD, AS THIS NUMBER IS THE SAME THAN THE TOTAL NUMBER OF LS FUNCTION FOR THE IMMEDIATE LOWER DIMENSION $N - 1$ (SEE ALSO THE TEXT ABOUT THE CONJECTURE FORMULATED IN THE PAPER.)

W	N=1	N=2	N=3	N=4	N=5
0	1	1	**1**	**1**	1
1	**2**	**4**	8	16	32
2	1	**4**	**12**	32	80
3	--	4	24	96	320
4	--	**1**	**14**	88	400
5	--	--	24	208	1120
6	--	--	12	192	1472
7			8	256	2560
8	--	--	1	**104**	1960
9	--	--	--	256	3520
10	--	--	--	192	4240
11	--	--	--	208	5920
12	--	--	--	88	5360
13				96	7360
14	--	--	--	32	6080
15	--	--	--	16	5920
16	--	--	--	1	**1882**
17	--	--	--	--	5920
18					6080
19	--	--	--	--	7360
20	--	--	--	--	5360
21					5920
22	--	--	--	--	4240
23	--	--	--	--	3520
24	--	--	--	--	1960
25	--	--	--	--	2560
26					1472
27	--	--	--	--	1120
28	--	--	--	--	400
29					320
30	--	--	--	--	80
31	--	--	--	--	32
32	--	--	--	--	1
Tot.	4	14	104	1882	94572

Next, we show that the mapping $t \mapsto t^*$ is a surjection: that is, that every balanced threshold function on $N+1$ variables equals t^* for some t. To do so, we show that every balanced threshold function has the property that, for all x, $f(x)$ and $f(\bar{x})$ are different. This will establish the result, because then, given any balanced threshold function f on $\{0,1\}^{N+1}$, f will equal t^* where $t \in T_N$ is the function given by $t(x) = f(x0)$. Now, suppose that $f \in T_{N-1}$ is balanced and that $f = [w, \theta]$ (where for no x do we have $\langle w, x \rangle = \theta$). Suppose, that, for some $x \in \{0,1\}^{N-1}$, $f(x) = f(\bar{x})$. Without loss of generality, suppose $f(x) = 1$ (a similar argument applying if $f(x) = 0$). Then

$$\langle w, x \rangle > \theta \text{ and } \langle w, \mathbf{1} - x \rangle > \theta.$$

That is,

$$\langle w, x \rangle > \theta \text{ and } \sum_{i=1}^{N+1} w_i - \langle w, x \rangle > \theta.$$

This implies that

$$\theta < \langle w, x \rangle < \sum_{i=1}^{N+1} w_i - \theta,$$

so

$$\sum_{i=1}^{N+1} w_i > 2\theta.$$

Now, suppose that v is any element of $\{0,1\}^{N+1}$ for which $f(v) = 0$. Then we have $\langle w, v \rangle < \theta$ and so

$$\langle w, \bar{v} \rangle = \langle w, \mathbf{1} - v \rangle = \sum_{i=1}^{N+1} w_i - \langle w, v \rangle > \sum_{i=1}^{N+1} w_i - \theta > \theta,$$

and hence $f(\bar{v}) = 1$. So, $f(v) = 0$ implies $f(\bar{v}) = 1$. So, the number of positive example in $\{0,1\}^{N+1} \setminus \{x, \bar{x}\}$ is at least $(2^{N+1} - 2)/1$ (at least half of them) and hence the total number of positive examples is at least $2 + (2^{N+1} - 2)/1 = 2^N + 1$ (when we include x and $\bar{x}$). But this contradicts the fact that f is balanced (which means the number of positive examples is exactly 2^N). The proof is now complete.

IV. The Computational Approach

We implemented a code that permit us to obtain straightforwardly all Boolean functions for up to N=4 variables without the need of checking for linearly separability. We create connected graphs of length 2^{N-1} starting from the null example. It can be easily shown that the total number of graphs with length 2^{N-1} is the double of the ones that includes always the null example, and thus we restrict our code to the construction of these graphs. All examples are ordered first according to their weight (number of ones) and for equal weight according to their binary(or equivalently any base representation) value. As said, starting from the null example to which we assign value 0, we create all possible extensions of the graph by adding a nearest neighbor (an example at Hamming distance 1). From all the new length-2 graphs, we consider the addition of a new example from all examples that are neighbors to the ones belonging to the graph but with the restriction that the new example have to be in a higer position in the order ranking to be elegible for the enlargement. This last requirement ensures that a same graph is not created twice. At each step positive unateness is checked and the graph is enlarged only if positive unateness is verified until the length of the graph is 2^{N-1}. A function f : $0,1^N \rightarrow 0,1$ is said to be unate if for every $i1, ..., n$ exactly one of the following holds: whenever the i-th bit is flipped from 0 to 1 then the value of f does not decrease (positive unateness) ; or whenever the i-th bit is flipped from 1 to 0 then the value of f does not decrease (negative unateness). Unateness is in a sense a more general form of monotonicity. It is well known that every threshold function is unate but there are some unate functions that are not threshold. It is also worth mentioning that opposite examples to the ones already in the graph are not considered are possible candidates for the enlargement as there are no balanced Boolean functions for which opposite examples have the same output value (see the demonstration of the theorem in previous section). We implemented the code for $N = 2, 3, 4$ and 5 variables and have been able to obtain straightforwardly all threshold functions in dimensions 1, 2, 3 and 4 respectively. We are currently improving the code to be able to compute higher dimension cases that require more computational and memory resources.

V. The Generalization Complexity of Linear Separable (Threshold) Boolean Functions

We analyze the complexity of the LS functions using a recently introduced measure related to the generalization ability that can be expected when the functions are implemented on a system capable of making predictions like Neural Networks, Decision Trees, Support Vector Machines (SVM), etc. It has also been shown that a significant correlation exists between the value obtained of the complexity of the functions and the number of configurations in the synaptic weight space of neural network implementing a given function, giving further confirmation of the potential of the complexity measure for predicting the generalization ability that can be obtained through the implementation of the functions. The complexity measure, named *generalization complexity* consists of two terms that measure the number of examples at Hamming distances 1 and 2 with opposite outputs ([12], [11]. The first term measures how sensitive the output of the function is in response to bit flippings and the second term essentially measures how symmetric is the function.

We compare the distribution of the complexity of the LS functions with values obtained for all the exhaustive set of functions for $N = 4$. It is expected that the complexity of the LS would be lower than for a general Boolean function as the LS functions can be implemented by a simple perceptron

[13]. In Fig. 6 the distribution of complexities of all the 65536 Boolean functions of 4 variables and for the 1882 linear separable ones is plotted. As expected, the mean complexity for the LS functions is lower than the mean for the whole set, 0.455 vs. 0.50. We have also computed the generalization ability obtained when these two sets of functions were implemented in neural networks architectures and also as expected from a set with lower complexity, the generalization ability was higher for the LS functions. The mean generalization ability for the LS functions was 0.598, while 0.5 was obtained for all the Boolean functions. The value of 0.5 for the whole sets of Boolean functions arise as a consequence of a No-Free-Lunch lemma demonstrated in [12].

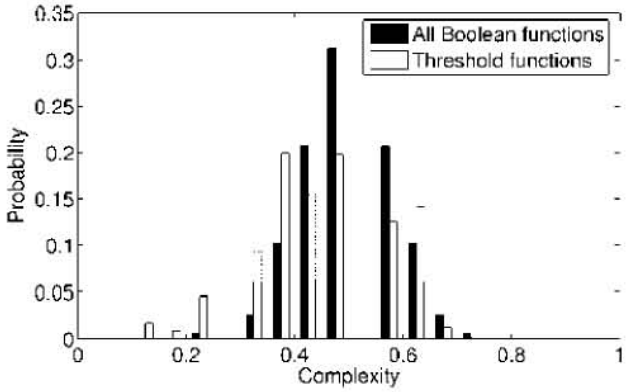

Fig. 6. The distribution of complexities for all and for the linear separable functions for $N = 4$ variables. The mean complexity of the functions is 0.5 and 0.455 for all the functions and for the LS ones respectively.

VI. Conclusions

A new constructive method for creating all linear separable functions have been developed. One of the key points of the method is that permits to obtain all threshold functions without needing to use a linear minimization method to check for their linear separability. The method rely on a theorem, that we proved, and some numerical evidence that showed that all Boolean threshold functions can be created straightforwardly. The procedure was checked up to N=5 and we are in the process to compute higher dimensions cases to see if consistent results are obtained. On the theoretical side, we are also investigating the relationship between balanced, threshold and unate functions to see if method can be supported by rigourously mathematical results. If the initial results are confirmed for higher dimensions (or in general) the algorithm will permit to check the linear separability of functions on N variables in a relative easy way by checking in dimension N+1 their balancedness and unateness.

Furthermore, the complexity of all the 1882 LS functions on $N = 4$ variables was analyzed and compared to the rest of the Boolean functions with 4 variables using a recently introduced measure of complexity. The results obtained are consistent with what is expected for the LS functions as their complexity is lower than of the rest of the functions but interestingly not too lower as might be expected. The generalization ability obtained from their implementation gives a confirmation of what was obtained using the complexity measure, essentially indicating that even if the threshold functions can be implemented by simple perceptron their generalization ability when trained with one hidden layer architectures, that can implement other functions, is not very large.

Acknowledgment

The authors acknowledge the support of Ministerio de Educación y Ciencia, España through grant number TIN2005-02984 (including FEDER funds). Leonardo Franco acknowledges support from the Ramón y Cajal programme from Ministerio de Educación y Ciencia, España.

References

[1] R.O. Winder. "Enumeration of Seven-argument Threshold Functions." *IEEE Transactions on Electronics Computers*. EC-14. pp. 315-325. 1965.

[2] S. Muroga. T. Tsuboi and C.R. Baugh. "Enumeration of Threshold Functions of Eight Variables. "*IEEE Transactions on Computers*. C-19. pp. 818-825. 1970.

[3] P.C. Ojha. "Enumeration of Linear Threshold Functions from the lattice of Hyperplane Intersections." *IEEE Transactions on Neural Networks*. 11. pp. 839-850. 2000.

[4] J. Zunic. "On Encoding and Enumerating Threshold Functions." *IEEE Transactions on Neural Networks*. 15. pp. 261-267. 2004.

[5] D.R. Smith. "Bounds on the Number of Threshold Functions." *IEEE Transactions on Electronic Computers*. EC-15. pp. 368-369. 1966.

[6] S. Yajima and T. Ibaraki. "A Lower Bound on the Number of Threshold Functions." *IEEE Transactions on Electronic Computers*. EC-14. pp. 926-929. 1965.

[7] D. Saad. "Capacity of the single-layer perceptron and minimal trajectory training algorithms." *J. Phys. A Math. Gen.* 26. pp. 3757-3773. 2003.

[8] W. Maass, G. Schnitger and E.D. Sontag. "A comparison of the computational power of sigmoid and Boolean threshold circuits." in Theoretical Advances in Neural Computation and Learning (Roychowdhury, V. P., Siu K. Y., and Orlitsky A., eds.). Kluwer Academic Publishers. pp. 127-151. 1994.

[9] V. Bohossian, P. Hasler, and J. Bruck. "Programmable neural logic," *IEEE Trans. Component, Packaging, Manuf. Technol. Part B*. 21. pp. 346 351. 1998.

[10] R. Zhang. P. Gupta. L. Zhong and N.K. Jha. "Threshold network synthesis and optimization and its application to nanotechnologies." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 24. pp. 107-118. 2005.

[11] L. Franco. "Generalization ability of Boolean functions implemented in feedforward neural networks," *Neurocomputing*. In Press. 2006.

[12] L. Franco and M. Anthony. "The influence of oppositely classified examples on the generalization ability of Boolean functions." *IEEE Transactions on Neural Networks*. In Press. 2006.

[13] J. Hertz, A. Palmer and R.G. Palmer. *Introduction to the theory of neural computing*. Reading. MA: Addison Wesley. 1991