Computational capabilities of feedforward neural networks: the role of the output function

José L. Subirats, Leonardo Franco, Iván Gomez and José M. Jerez

Departamento de Lenguajes y Ciencias de la Computación
E.T.S.I. Informática, Universidad de Málaga
Campus de Teatinos, s/n
29071 Málaga, Spain

Abstract. We present an analysis of the computational capabilities of feed-forward neural networks focusing on the role of the output function. The space of configurations that implement a given target function is analyzed for small size networks when different output functions are considered. The generalization complexity and other relevant properties for some complex and useful linearly separable functions are also analyzed. The results indicate that efficient output functions are those with a similar Hamming weight as the target output that at the same time have a high complexity.

1 Introduction

Feed-forward neural networks are one of the most used methods for prediction and classification tasks and have been applied to several problems in a wide range of areas including secondary structure protein prediction, stock market prediction, pattern recognition tasks, medical prognosis and diagnosis, etc. [1]. Feed-forward neural networks can be trained from examples without knowing particular details of the problem under consideration and their success has been that the generalization ability obtained in most cases tends to be relatively good. How many hidden layers and how many nodes to include in a neural network architecture is a problem that has not been yet clearly answered, despite the fact that some regularization techniques can alleviate the problem of architecture selection [2] .

The issue about the computational capabilities of feed-forward neural networks has been mainly analyzed indirectly through the calculation of some properties of certain size architectures. Exhaustive studies have not been carried out much because of the computational complexity involved. For example, the introduction of the VC dimension theory has helped to partially understand the issue of generalization in a general framework but has not clarified practical implementation issues [1, 3]. In this work, we analyze the computational capabilities of some reduced size neural architectures focusing on the role of the output function. Besides the general interest on knowing which functions a certain architecture is able to implement, an extra interest of the present study is related to constructive methods for neural networks. Constructive algorithms create neural architectures specifically for the problem on analysis starting from small architectures and adding successively neurons and connections, and different strategies for evolving the architectures have been designed [4–6]. Knowing which functions are best suited for the role of output function would be and advantage at the time of the architecture selection and learning processes, as it can help on the design of new and more efficient methods. Works related to the present study, includes [7] where the computational capabilities of restricted two layer perceptrons was analyzed. Also in Ref. [8] the volume of space occupied by continuous weights implementing a given function was computed together with entropy values for small sizes architectures. The study of the computational capabilities of neural networks is also related to the study of the circuit complexity

[9, 10], that is also linked to the practical implementation of logic circuits using threshold logic.

2 The importance of the weight of the target function in the selection of the output

We run numerical simulations in small size 3,4 and 5 input variables architectures using Boolean functions as target functions. The case N=3 has been analyzed exhaustively, analyzing all the existing configurations for all different output functions using up to 3 neurons in a single hidden layer architecture that is enough to compute any of the 152 non-linearly separable function. For networks with 4 variables all 65536 Boolean functions have been tested for some specific choice of the output function.

From the exhaustive study for all the 152 non-linearly separable functions of 3 variables, a 3 hidden node single layer architecture was used. One of the results of the study was that there exists a strong correlation between the number of bits ON (number of 1's or equivalently the Hamming weight) in the truth vector of the Target and Output functions. The target functions were all the 152 non linearly separable functions while the output function for the correlation analysis was one of the best output functions found. We measured the Pearson correlation coefficient between the number of 1's in both truth vectors obtaining a value of R=0.8384 (associated p-value less than 1E-8). The number of pairs analyzed to obtain this correlation coefficient was 2354 instead of the 256 cases because in many cases more than an single optimal Output functions exists. Also it was observed that the generalization complexity of the best output functions was quite high in comparison to the rest of the linearly separable functions.

The output function in a neural network is normally a linear separable function. However by including an extra layer of nodes, the analysis can be extended to include output functions that are non-linearly separable. We analyzed a particular case in which the output neuron considered together with the second layer of nodes implement the well known parity function [13]. Besides from the fact that the parity function is a very complex function, the motivation for using and studying it, was that we are currently developing a constructive algorithm that works very efficiently using the parity function as output. The parity is not a threshold function but many implementations using threshold functions are known. In Fig. 1 we show a standard architecture that can be used for compute an arbitrary target function. Regarding these kind of networks composed of two hidden layers of neurons we note that there are some results showing that they are very efficient for obtaining generalization [3].

From the analysis of the best output functions we also noticed that the majority function is also a very good output function. The majority boolean function is defined as a function whose output is active only when half or more than half of the Ninput bits are active. The majority function has the advantage that can be implemented in threshold circuits using only weights equal to 1 and a threshold of $Int(N/2)$.

We show in Fig. 2 the number of possible solutions for all the 152 non-linearly separable functions of 3 variables for the case of three output functions, the parity function (top), the majority function (middle) and the case depicted at the bottom of Fig. 2 corresponds to the choice of the best linearly separable function as output function. For this last case, we analyze the exhaustive results and pick the function for which the maximum number of configurations implementing the desired function exist. We compute as well the entropy of the different architectures in which the output function was fixed. The entropy of an architecture is defined as:

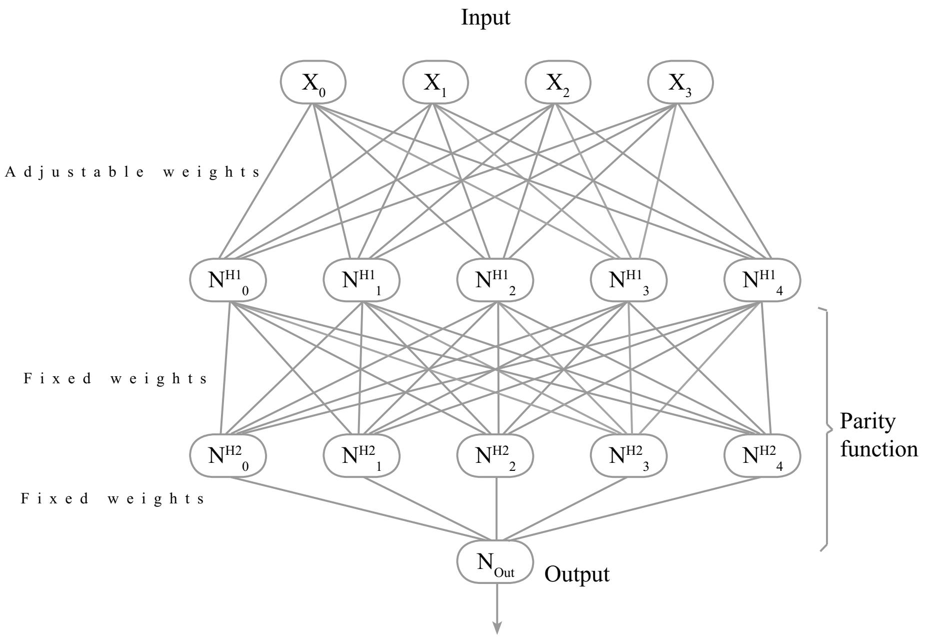

Fig. 1. Structure of a two hidden layer network architecture used for analyzing the computational capabilities of networks using the parity function as output.

$$S = \sum_{configurations} - p_i \log_2 p_i, \tag{1}$$

where p_i is the probability that a given function would be computed by a certain configuration. A large value of the entropy indicates that the space of configurations is distributed almost uniformly for the different functions that can be implemented and also gives a relative indication on the number of different functions that the network is able to implement. For the parity function the value of the entropy found was 7.99, almost equal to the maximum possible value of 8 ($\log_2(256)$) indicating that when the parity function is used as output there always exist a configuration of the internal nodes that permit to implement any desired function. The majority function was the best output function among the set of non-linearly separable functions. In the figure is also shown the number of configurations that implement each of the non-linearly separable functions by the best linearly separable function.

The simulations results for the case of N=4 variables carried with a fixed output function showed that within the linearly separable functions, the majority function is quite efficient for implementing a given target function. We also run simulations fixing the output function in an exhaustive algorithm introduced in [14] and the results with N=5 and 6 inputs confirm that majority functions are quite efficient as output functions.

3 The generalization complexity of some efficient output functions

The generalization complexity measure has been introduced in Refs. [11, 12] as a measure of the complexity of a Boolean function in terms of the generalization ability that can be obtained when the function is implemented in a neural architecture. In this section, the complexity is analyzed analytically for the majority function and also for the parity function. The generalization complexity measure contains two terms C_1 and C_2 that measures the number of pairs at Hamming distances 1 and 2 having opposite output. The case of the parity function is quite easy to analyze because is a symmetric function for which all neighboring examples have opposite output. The value of first term of the parity function is 1 (maximum value) for any number of input variables, while the second term is 0.

We analyze the case of a majority function with N inputs considering N even as this will make the analysis simpler. The case with N odd can be treated in analogous form but does not add anything new to the analysis and then is skipped from the paper. To perform the analysis we consider the partially ordered set (poset) because the input instances are ordered according to their Hamming weight and the diagram will help to follow the calculations. The majority function is a threshold function, and thus there exist a plane separating the positive (true) and negative (false) examples; and the examples at both sides of this plane are those that contribute to the first term, C_1 of the complexity measure. In the poset of Fig. 3 where the majority function on 4 variables is shown, it is easy to see which examples will contribute to the computation of the complexity measure. In the figure, examples with output 1 and 0 are shadowed with different colors but also neighboring examples are linked by straight lines. The first term of the complexity measure is then proportional to the number of linked pairs with different output, and from the figure is easy to see that only two layers of examples will contribute to the measure. These two layers are the second one from the top, containing examples with Hamming weight 1 and the middle layer containing examples with Hamming weight 2. We can see that each of the 4 examples with Hamming weight equal to 1 have 3 neighboring examples in the middle layer and then the total number of opposite pairs is 12. As in dimension 4, each of the 16 examples has 4 nearest neighbors, the total number of pairs would be $24 = 16 \times 4 \times 0.5$ resulting in a value of C_1 equal to 0.5.

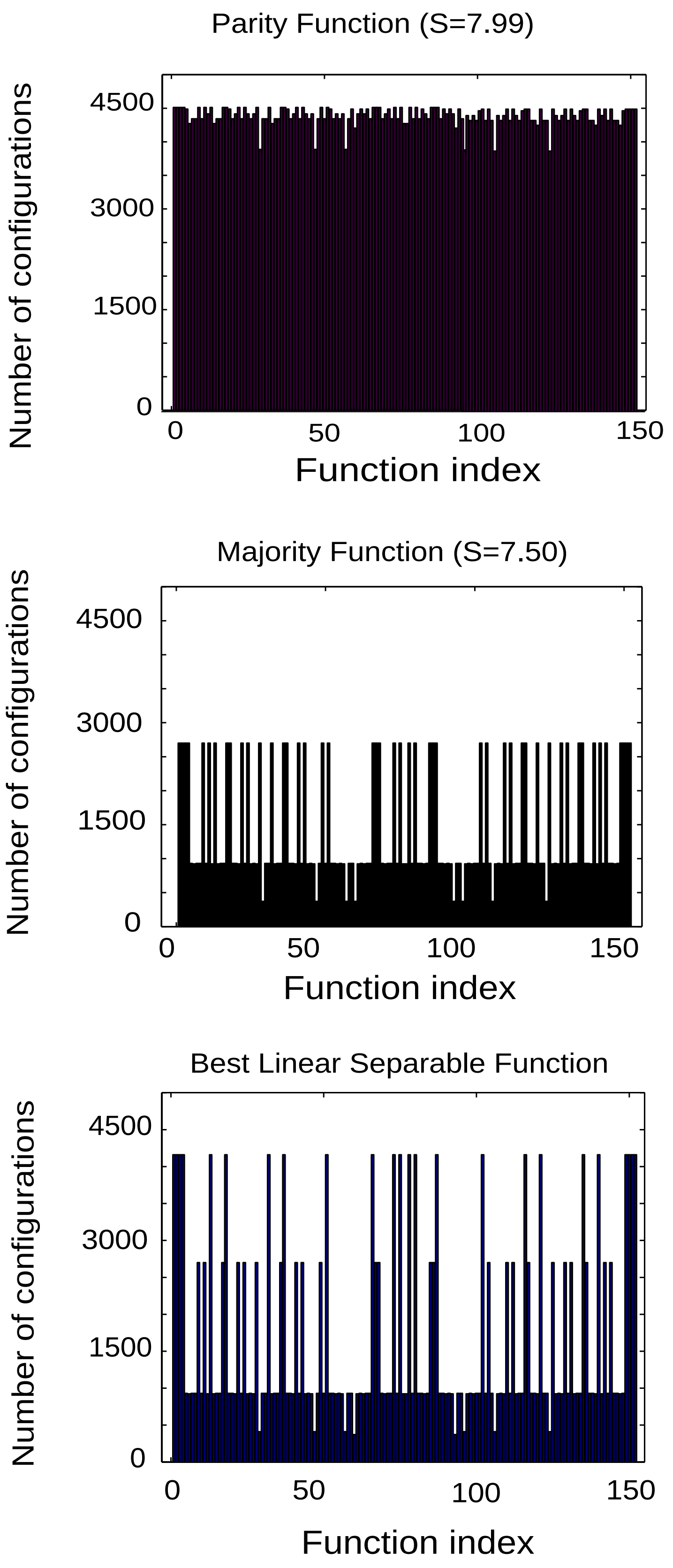

Fig. 2. The number of possible solutions for all the non-linearly separable Boolean functions of 3 variables implemented in three hidden nodes architecture for three different output functions. The results for the parity function are shown on top, for the majority function in the middle and the bottom case correspond to the best linear separable function.

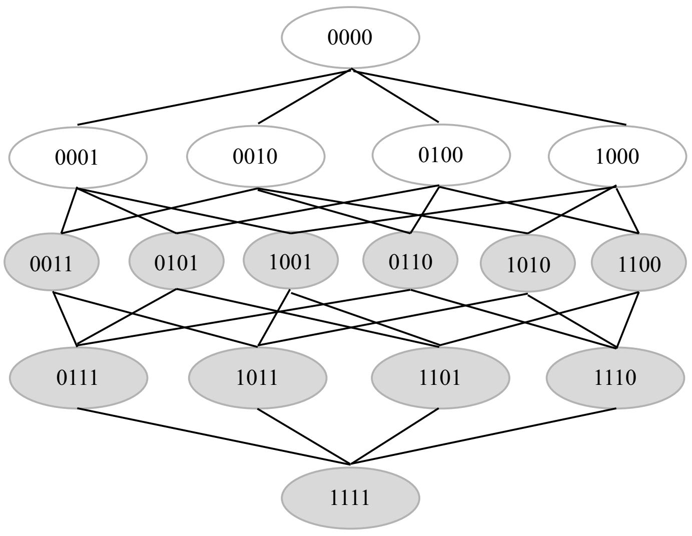

Fig. 3. The partially ordered set (POSET) for the majority function on 4 variables. In each of the 5 levels the examples are organized according to the Hamming weight (number of 1's). The examples whose output is 1 are grey colored.

The general case of N variables is now analyzed, noting that also only two adjacent levels are involved in the calculation. One of this two levels is always the middle layer that contains $\binom{N}{\frac{N}{2}}$ examples. Each example have $\frac{N}{2}$ neighbors in the neighboring level located in the opposite region, and then the total number of neighboring pairs of examples with opposite output is $\binom{N}{\frac{N}{2}} \times \frac{N}{2}$. By dividing this number by the total number of pairs we obtain the value of the first term of the generalization complexity for the majority function:

$$C_1 = \frac{\binom{N}{\frac{N}{2}} \times \frac{N}{2}}{2^N \times \frac{N}{2}} = \frac{\binom{N}{\frac{N}{2}}}{2^N}$$

that for large values of N becomes:

$$C_1 \sim \frac{2^N}{\sqrt{N}} \times \frac{1}{2^N} = \frac{1}{\sqrt{N}}$$

The value of the term C_2 can be carried in a similar way to obtain that its value its exactly the same of the first term. The calculations are skipped due to space restrictions. As a comparison of the complexity of the majority function, we also considered the comparison function that is also a linearly separable Boolean function. The comparison function it is known to be one of the most complex linearly separable functions as it implementation require exponentially large weights. The comparison function is defined as the function that gives a 1 if the first half of the bits considered as a separate number is larger than the second part. For example, an input with 6 variables 110100 will have an output equal to 1 because the first half 110 is larger than the second 100. In table 1 a comparison of the complexity for the three analyzed functions is shown for up to $N = 12$.

	Majority		Comparison		Parity	
	C_1	C_2	C_1	C_2	C_1	C_2
2	0.5	0.5	0.5	0	1	0
4	0.375	0.375	0.375	0.375	1	0
6	0.312	0.312	0.292	0.292	1	0
8	0.273	0.273	0.234	0.234	1	0
10	0.246	0.246	0.194	0.194	1	0
12	0.226	0.226	0.164	0.164	1	0
N	$\frac{1}{\sqrt{N}}$	$\frac{1}{\sqrt{N}}$	$--$	$---$	1	0

Table 1. The value of the complexity terms C_1 and C_2 for the Majority, Comparison and Parity functions for different values of the number of input variables.

4 Conclusions

We have analyzed the computational capabilities of small size feed-forward neural architectures analyzing the importance of the choice of the output function. The results indicates that good output functions are those that have a Hamming weight similar to the one of the target function but are also functions that are quite complex. We found that the majority function is an efficient function and analyzed its complexity for large values of variables. As a comparison, we analyzed the complexity of the comparison function, considered to be one of the most complex function belonging to the linearly separable set of functions. It turned out that the complexity of the comparison function decreases faster than the complexity of the majority function. The analysis of the majority function for large values of N shows that its complexity decays as the square root of N, and this might indicate that as N grows its efficiency as output function might decrease. We have also considered the case of using the parity function as output function. The analysis shows that it might be a very efficient output function, as optimal entropy was obtained from the calculations in small size networks, and also because its complexity is quite large and does not decrease as N grows. The final conclusion of our study could be that within the set of linearly separable functions the majority function is a quite efficient output functions, but it seems a better strategy to use a second layer of nodes and implement a function like the parity as output of the network.

5 Acknowledgements

The authors acknowledge support from CICYT (Spain) through grant TIN2005-02984 (including FEDER funds). LF acknowledges support from the Spanish Ministry of Education and Science through a Ramón y Cajal fellowship.

References

1. Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation.* Macmillan, IEEE Press.
2. Caruana, R., Lawrence, S., and Giles, C.L. (2001). Overfitting in Neural Networks: Backpropagation, Conjugate Gradient, and Early Stopping. In: *Advances in Neural Information Processing Systems, MIT Press, 13*, pp. 402-408.
3. Franco, L., Jerez, J.M., and Bravo-Montoya, J.M. (2005). Role of Function Complexity and Network Size in the Generalization Ability of Feedforward Networks. *Lecture Notes in Computer Science, 3512*, pp. 1–8.
4. Mezard, M. and Nadal, J.P. (1989). Learning in feedforward layered networks: The tiling algorithm, J. Physics A., 22, pp. 2191–2204.
5. Frean, M. (1990). The upstart algorithm: A method for constructing and training feedforward neural networks, Neural Computation, 2, pp. 198–209.
6. Keibek, S.A.J., Barkema, G. T., Andree, H.M.A., Savenlie, M.H.F. and Taal, A. (1992). A fast partitioning algorithm and a comparison of binary feedforward neural networks. Europhys. Lett., 18, pp. 555–559.
7. Priel, A., Blatt, M. Grossman, T., Domany, E., and Kanter, I. (1994). Computational Capabilities of restricted two layered perceptrons, *Phys. Rev. E., 50*, 577.
8. Sprinkhuizen-Kuyper, I.G., and Boers, E.J.W. (1996). Probabilities and entropy of some small neural networks for boolean functions. Technical report 96-30, Dept. of Computer Science, Leiden University, Holland.
9. Siu, K.Y., Roychowdhury, V.P., and Kailath, T. (1991). Depth-Size Tradeoffs for Neural Computation *IEEE Transactions on Computers*, *40*, pp. 1402-1412.
10. Parberry, I. (1994). *Circuit Complexity and Neural Networks.* MIT Press, Cambridge, MA.
11. Franco, L. (2006). Generalization ability of Boolean functions implemented in feedforward neural networks. *Neurocomputing, 70*, pp. 351-361.
12. Franco, L. and Anthony, M. (2006). The influence of oppositely classified examples on the generalization complexity of Boolean functions. *IEEE Transactions on Neural Networks. 17*, pp. 578–590.
13. Franco, L. and Cannas, S.A. (2001). Generalization Properties of Modular Networks: Implementing the Parity Function. *IEEE Transactions on Neural Networks*, *12*, pp. 1306–1313.
14. Subirats, J.L., Gòmez, I., Jerez, J.M. and Franco, L. (2006). Optimal Synthesis of Boolean Functions by Threshold Functions. *Lecture Notes on Computer Science, 4131*, pp. 983-992.