Role of Function Complexity and Network Size in the Generalization Ability of Feedforward Networks

Leonardo Franco[1,2], José M. Jerez-Aragonés[2], and José M. Bravo Montoya[2]

[1] Dept. of Experimental Psychology, University of Oxford,
South Parks Road, Oxford OX1 3UD, UK
Leonardo.Franco@psy.ox.ac.uk
[2] Departamento de Lenguajes y Ciencias de la Computación,
University of Málaga, 29071 Málaga, Spain

Abstract. The generalization ability of different sizes architectures with one and two hidden layers trained with backpropagation combined with early stopping have been analyzed. The dependence of the generalization process on the complexity of the function being implemented is studied using a recently introduced measure for the complexity of Boolean functions. For a whole set of Boolean symmetric functions it is found that large neural networks have a better generalization ability on a large complexity range of the functions in comparison to smaller ones and also that the introduction of a small second hidden layer of neurons further improves the generalization ability for very complex functions. Quasi-random generated Boolean functions were also analyzed and we found that in this case the generalization ability shows small variability across different network sizes both with one and two hidden layer network architectures.

1 Introduction

Neural networks are nowadays widely use in different applications in pattern recognition and classification tasks due to their ability to learn from examples and to generalize. There exists some general theoretical results about the size of the network needed to implement a desired function [1, 2, 3] but at the time of the implementation the theory is not always accurate. In practice, the problem of selecting a network architecture for a determined application is mainly based on the "trial-and-error" method as no theoretical formula gives clear insight into this problem. Answers to general questions as: Are two hidden layer networks better than one ? or Does larger networks generalize better than smaller ones? are still controversial. A simple and general idea about what network size utilize comes from Occam's razor: the simpler the solution the better, but it has been shown that very large networks perform sometimes better than smaller ones [3, 4]. The reasons for the previous findings are still unclear and it seems that they arise from properties of the backpropagation algorithm combined with validation

J. Cabestany, A. Prieto, and D.F. Sandoval (Eds.): IWANN 2005, LNCS 3512, pp. 1–8, 2005.
© Springer-Verlag Berlin Heidelberg 2005

procedures that avoid overfitting [1, 4, 5, 6, 7]. Another results [8] state that what matters to obtain valid generalization is the value of the weights and not the size of the network but this does not solve completely the size dilemma. Moreover, most of the practical results based on simulations concentrate on the use of very few functions, even the most complete studies use less than 30 different functions [6, 7], and also in general the complexity of the analyzed functions is ignored [3, 9].

The complexity of Boolean functions has been studied for a long time, the aim of that being to have a criterion for deciding if a problem is easier to solve or implement than another [10]. Within the area of circuit complexity the complexity of a Boolean function has been defined as the minimum size of a circuit that could implement it, while the size of the circuit is measured as the number of nodes that composed the circuit. Many results have been derived for certain classes of functions, as for symmetric and arithmetic ones, where in general, bounds on the size of the circuits to compute the functions are obtained [10, 11], but all these measures are quite complicate to compute in most cases. In [3], functions were generated with different complexity by selecting a parameter K that controlled the size of the maximum weight in the network. They showed that functions with higher K were more complex, and expected the generalization to be worse as K increases. In this paper, we use a recently introduced measure for the complexity of Boolean functions [9, 12] to analyze how the network architecture affects the generalization ability obtained for different classes of functions grouped according to their complexity.

2 A Measure for the Complexity of Boolean Functions

The measure proposed in [9] is based on the results obtained in [13, 14], where a close relationship between the number of examples needed to obtain valid generalization and the number of neighbouring examples with different outputs was found. The complexity measure $\mathcal{C}[f]$ is obtained from the number of pairs of neighbouring examples having different outputs. The complexity measure used in this paper is defined as:

$$\mathcal{C}[f] \;=\; \mathcal{C}_1[f] + \mathcal{C}_2[f], \tag{1}$$

where $\mathcal{C}_i[f]$, $i = 1, 2$ are the terms taking into account pairs of examples at a Hamming distance one and two. The first term can be written as:

$$\mathcal{C}_1[f] = \frac{1}{N_{ex} * N_{neigh}} \sum_{j=1}^{N_{ex}} \left(\sum_{\{l|Hamming(e_j,e_l)=1\}} (|f(e_j) - f(e_l)|) \right), \tag{2}$$

where the first factor, $\frac{1}{N_{ex} * N_{neigh}}$, is a normalization one, counting for the total number of pairs considered, N_{ex} is the total number of examples equals to 2^N, and N_{neigh} stands for the number of neighbour examples at a Hamming distance

of 1. The second term $\mathcal{C}_2[f]$ is constructed in an analogous way. The complexity of the Boolean functions using the measure of Eq. 1 ranges from 0 to 1.5 [9].

3 Simulations Results

To analyze the generalization ability of different architectures and to study how this property change with network size we carried intensive numerical simulations for symmetric and quasi-random generated Boolean functions. All the networks were trained with Backpropagation with momentum. An early stopping procedure was implemented with the aim of avoid overtraining and improve generalization. The validation error was monitored during training and at the end of the maximum number of epochs permitted, the generalization error was taken at the point where the validation error was minimum. A important use of the complexity measure is that permit to analyzed the complexity of the output functions and compared it to the complexity of the target functions, to analyze what functions the networks generalize to. The number of examples available for training was 256 as in all cases we consider input functions of size N=8. We divided the examples in a training set containing 128 examples, and into validation and generalization test sets containing 64 examples each one. The learning rate constant was fixed during training and equal to 0.05 and the momentum term was set to 0.3. The maximum number of epochs allowed for training was selected after some initial tests and was selected to be approximated 4 to 5 times larger than the optimal training time (typical values were 600 to 1000). Two sets of functions symmetric and quasi-random functions generated starting from the parity function were considered and the results are described in the subsections below. To analyze the dependence of the generalization ability with the complexity of the functions the functions were grouped in ten different levels of complexity from 0 to 1.0.

3.1 Symmetric Functions

An important class of Boolean functions are the symmetric ones, those with values independent of the order of the input, i.e., the output of the examples depends only on the total number of input bits ON (the number of 1's). The class of symmetric functions contains many fundamental functions like sorting and all types of counting ones, including also the well known parity function [9, 10, 11]. They have been extensively studied and many lower bounds have been obtained for circuits computing them. A general result states that a circuit of size $\mathcal{O}(\sqrt{N})$ gates and depth 3 with polynomial weights is enough to compute all the symmetric functions [11].

We performed simulations using all 512 existing symmetric functions for the case $N = 8$, using neural networks with one and two hidden layers. The number of neurons for the analyzed cases in which the networks have only one hidden layer was NH1=$5, 10, 25, 50, 250, 500$. The best result, in terms of the generalization ability obtained, was by using the architecture with NH1=250. In Fig. 1 the generalization ability as a function of the function complexity is shown for

the different networks utilized. The results were computed from five averages taken for every symmetric function. Standard deviations were also computed for every complexity group and it showed that the differences obtained for different architectures (for example between the network with NH1=250 NH2=0 and the architecture with NH1=250 NH2=10) were statistically significant for groups of functions with the same complexity. We also computed the average absolute value of the weights in the different layers and the result was that the size of the weights was proportional to the training time, that was also found to be proportional to the function complexity (In [3, 8] similar findings were obtained).

The complexity of the functions to which the networks generalize to was also investigated (i.e. the functions that the trained networks compute at the point of minimum validation error). We found that in almost all cases the output functions were more complex than the target ones.

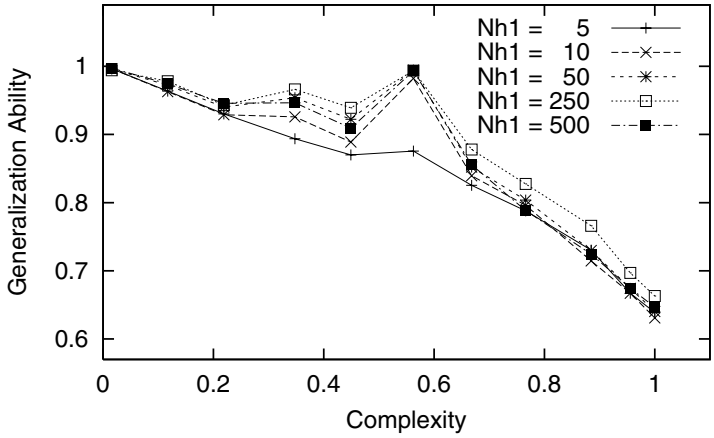

Fig. 1. Generalization ability vs function complexity for all Symmetric Boolean functions with N=8 inputs implemented on single hidden layer networks with a number of hidden neurons NH1= 5,10,25,100,250 and 500

We also analyzed the generalization ability for cases of architectures having two hidden layers of neurons, first by restricting our study to networks with NH1=250, that was the optimal value found for one hidden layer networks. The number of second hidden neurons analyzed were NH2=5, 10, 25, 50, 250, 500. The best results were obtained with an architecture 8-250-10-1, that outperformed the architecture 8-250-1. In Fig. 2 the results are shown only for some two-layer architectures (for clarity) in comparison to the best one hidden layer architecture. A further enlargement of the second layer of neurons did not improve the generalization ability. Other cases with a different number of neurons in the first and second layer were considered but the results were worse than the cases already mentioned.

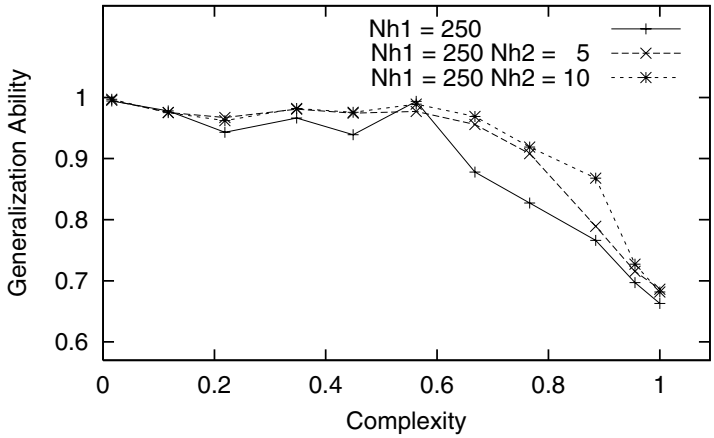

Fig. 2. Generalization ability vs function complexity for all Symmetric Boolean functions with N=8 inputs for the cases of two two hidden layer architectures with 250 in the first hidden layer and NH2=5,10 in the second hidden layer. For comparison the results for the case of having a single layer with NH1=250 is also shown

Table 1. Average generalization ability, training time (epochs) and final training error obtained for different size architectures with one or two hidden layers of neurons constructed to compute all Boolean symmetric functions with N=8 inputs

Neurons in 1^{st} hidden layer	Neurons in 2^{nd} hidden layer	Generalization ability	Train. time (epochs)	Final train. error
5	–	0.761	256	0.111
50	–	0.781	167	0.019
250	–	0.803	132	0.060
500	–	0.780	291	0.187
250	5	0.839	192	0.150
250	10	**0.852**	186	0.136
250	25	0.846	159	0.142

It is possible to observe from Fig. 1 that for networks with a single layer of neurons the use of a large number of neurons up to 250 increases the generalization ability for the whole ranges of complexities. When a second layer of weights is included, the main effect was that the network improved the generalization ability on functions with a complexity larger than 0.6 (See Fig. 2). In table 1, the average generalization ability obtained for some of the architectures used is shown together with the training time (in epochs) until the point in which the minimum validation error was found and also is shown the training error at the end of the training procedure to give an idea of the computational capacity of the network. In a previous study [9], in which the architecture sizes were much restricted, up to 30 neurons in a single hidden layer, we found only a slightly im-

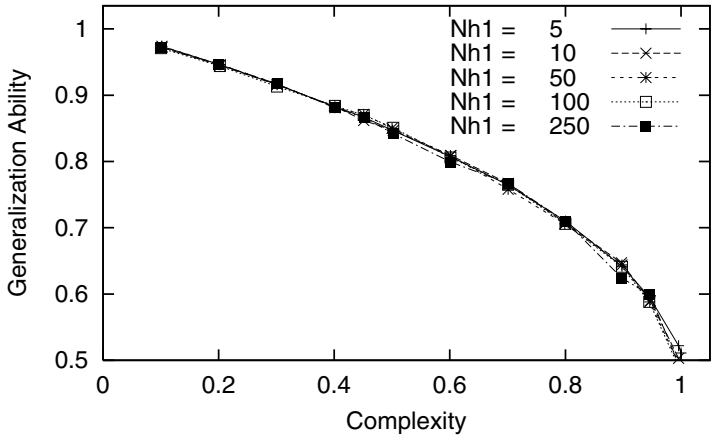

Fig. 3. Random functions implemented on a one hidden layer architecture

Table 2. Average generalization ability, training time (epochs) and final training error obtained for different size architectures with one or two hidden layers of neurons constructed to compute quasi-random Boolean functions generated by modifications of the parity function for N=8 inputs

Neurons in 1^{st} hidden layer	Neurons in 2^{nd} hidden layer	Generalization ability	Train. time (epochs)	Final train. error
5	–	0.789	79	0.063
50	–	0.787	58	0.004
250	–	0.785	132	0.060
5	5	**0.798**	19	0.165
5	10	0.795	44	0.136
5	50	0.792	83	0.129

provement in the generalization ability for the case of very low complex functions and thought that the effect could be explained on the basis that larger networks have more configurations implementing trivial functions. Whether the previous assertion is true is worth but difficult to elucidate and in terms of the generalization ability that can be obtained using Backpropagation combined with early stopping it does not seem to be the case at least for symmetric functions.

3.2 Random Functions

There exists 2^{2^N} Boolean functions of N inputs, making their general study very complicated except for very simple and small cases. Totally random functions are very complex with an average complexity around 1.0 [9].

To analyze a large set of different functions we generate functions with different complexity by modifying with a certain probability the outputs equal to 1 of the parity function, to obtain functions that are asymmetric in the number

of outputs and with a complexity in the range 1.0 to 0. One hundred functions were analyzed for each of 10 levels of complexities in which the functions where grouped, with average complexity from 0.1 to 1.0 in steps of 0.1. We run simulations using networks with a single hidden layer and a number of hidden neurons NH1 equal to NH1=5,10,25,50,100,250,500 and obtained that the generalization ability for the different complexity groups of functions did not change much with the size of the hidden layer, as can be appreciated from Fig. 3. On average the best one hidden layer architecture was the one with only NH1=5 neurons. It was also obtained as in the previous section that the introduction of a second hidden layer of neurons improve the overall generalization ability and the optimal values found were NH1=5 and NH2=5. In table 2 average results are shown for some of the architectures used. Note from the results in table 2 that the final training error was higher for the optimal architectures, and this might indicate that larger architectures has a major propensity to overfitting for these quasi-random functions while two hidden layer networks seems to suffer less of this problem that the one hidden layer ones.

4 Discussion

We have analyzed the generalization ability of different network architectures studying how the generalization ability is affected by the complexity of the functions being implemented and by the size of the architecture. It is generally assumed that the generalization ability depends on the complexity of the target function but we do not know of previous studies addressing this point except from [3], and we think that this is mainly due to the lack of a simple measure for the complexity of the functions. The use of a recently introduced complexity measure permit us to have a clearer picture about how different architectures perform by permitting the functions under analysis to be grouped according to their complexity [9].

We obtained that large neural networks do not overfit much if trained with the Backpropagation algorithm combined with an early stopping procedure as it was observed in previous studies [3, 4]. For the case of network architectures with a single layer of hidden neurons, the optimum value for the number of neurons was obtained with NH1=250 when the set of symmetric Boolean functions was considered. It was further shown that generalization ability improved if a second layer of hidden neurons was used where the optimal number of neurons found was NH2=10. In particular, it was observed that the introduction of the second layer of neurons improves the generalization ability of very complex symmetric functions with a complexity between 0.6 and 0.9.

For the case where quasi-random functions were analyzed it was found that the optimal network with a single hidden layer was very small NH1=5, but when using larger networks of up to NH1=500 the average generalization values were quite similar (See table 2). It was also observed in this case that the introduction of a second layer of hidden neurons further improved the average generalization

ability, but no significant changes were observed in terms of the functions with different complexity.

We are currently extending the measure of complexity to continuous input functions to be able to carry out similar studies with real problems.

References

1. Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Macmillan/IEEE Press.
2. Baum, E.B. & Haussler, D. (1989) What size net gives valid generalization ? *Neural Computation*, *1*, pp. 151-160..
3. Lawrence, S., Giles, C. L., & Tsoi, A. C. (1996). What Size Neural Network Gives Optimal Generalization ? Convergence Properties of Backpropagation. In Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced Computer Studies, Univ. of Maryland.
4. Caruana, R., Lawrence, S., & Giles, C.L. (2001). Overfitting in Neural Networks: Backpropagation, Conjugate Gradient, and Early Stopping. In Leen, T. K., Dietterich, T. G. & Tresp, V. editors, Advances in Neural Information Processing Systems, MIT Press, *13*, pp. 402-408.
5. Krogh, A. & Hertz,J.A. (1992) A simple weight decay can improve generalization. In J.E. Moody, S. J. Hanson, & R. P. Lippmann editors, Advances in Neural Information Processing Systems Morgan Kaufmann, San Mateo, CA, *4*, pp. 950 957.
6. Prechelt, L. (1998). Automatic Early Stopping Using Cross Validation: Quantifying the Criteria. *Neural Networks*, *11*, pp.761-767.
7. Setiono,R. (2001) Feedforward neural network construction using cross-validation, *Neural Computation*, *13*, pp. 2865-2877.
8. Bartlett,P.L. (1997). For valid generalization the size of the weights is more important than the size of the network. In M.C. Mozer, M. I. Jordan, & T. Petsche, editors, Advances in Neural Information Processing Systems, MIT Press, *9*, pp. 134-140 .
9. Franco, L. & Anthony, M. (2004). On a generalization complexity measure for Boolean functions. In *Proceedings of the 2004 IEEE International Joint Conference on Neural Networks*, IEEE Press, pp. 973-978.
10. Wegener, I. (1987) *The complexity of Boolean functions*. Wiley and Sons Inc.
11. Siu, K.Y., Roychowdhury, V.P., & Kailath, T. (1991) Depth-Size Tradeoffs for Neural Computation *IEEE Transactions on Computers*, *40*, pp. 1402-1412.
12. Franco, L. & Cannas, S.A. (2004). Non glassy ground-state in a long-range antiferromagnetic frustrated model in the hypercubic cell *Physica A*, *332*, pp. 337-348.
13. Franco, L. & Cannas, S.A. (2000). Generalization and Selection of Examples in Feedforward Neural Networks. *Neural Computation*, *12*, 10, pp. 2405-2426.
14. Franco, L. & Cannas, S.A. (2001). Generalization Properties of Modular Networks: Implementing the Parity Function. *IEEE Transactions on Neural Networks*, *12*, pp. 1306-1313.