Neural Process Lett (2009) 30:71–87
DOI 10.1007/s11063-009-9108-2

Neural Network Architecture Selection: Can Function Complexity Help?

Iván Gómez · Leonardo Franco · José M. Jerez

Published online: 15 July 2009
© Springer Science+Business Media, LLC. 2009

Abstract This work analyzes the problem of selecting an adequate neural network architecture for a given function, comparing existing approaches and introducing a new one based on the use of the complexity of the function under analysis. Numerical simulations using a large set of Boolean functions are carried out and a comparative analysis of the results is done according to the architectures that the different techniques suggest and based on the generalization ability obtained in each case. The results show that a procedure that utilizes the complexity of the function can help to achieve almost optimal results despite the fact that some variability exists for the generalization ability of similar complexity classes of functions.

Keywords Generalization ability · Neural network · Network architecture · Boolean functions · Complexity

1 Introduction

Feed-forward neural networks trained by back-propagation have become a standard technique for classification and prediction tasks given their good generalization properties. However, the process of selecting an adequate neural network architecture for a given problem is still a controversial issue.

Several theoretical bounds for the number of hidden neurons necessary to implement a given function have been derived using different methods and approximations. Baum and

I. Gómez · L. Franco (✉) · J. M. Jerez
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga,
Campus de Teatinos S/N, 29071 Málaga, Spain
e-mail: lfranco@lcc.uma.es

I. Gómez
e-mail: ivan@lcc.uma.es

J. M. Jerez
e-mail: jja@lcc.uma.es

colleagues [4] have obtained some bounds on the number of neurons in an architecture related to the number of training examples that can be learnt using networks composed of linear threshold networks. An important contribution about the approximation capabilities of feed-forward networks was made by Barron [3], who computed an estimation of the number of hidden nodes necessary to optimize the approximation error. Camargo et al. [6] obtained a result for estimating the number of nodes needed to implement a function using Chebyshev polinomials and using previous results from Scarselli et al. [30] about the number of nodes needed for approximating a given function by polynomials. It is worth noting that some of the previous bounds and other existing results have been obtained using the formalism of the Vapnik and Chervonenkis theory, that have helped much to understand the process of generalization in neural networks [5,17]. Singular value decomposition [34], geometrical techniques [1,26,39], information entropy [38] and the signal-to-noise-ratio [22] have been also used to analyze the issue about the optimal number of neurons in a neural architecture. The previous mentioned theoretical results have helped much to understand certain issues regarding the properties of feed-forward neural networks but unfortunately at the time of practical implementations the bounds are loose or difficult to compute. This fact has lead to more practical approaches where researchers investigated the problem of choosing a proper neural architecture through numerical simulations. For example, Lawrence et al. [18] have shown that the backpropagation algorithm can work relatively well if a large network is chosen and the size of the weights is constrained with regularization techniques such as weight decay [17]. Even if this is the case, still a size has to be chosen and the smaller the architecture the better, specially given the longer training times required for the learning process. Given this situation and to alleviate the computational burden related to the trial-and-error method normally used, different researchers and companies have came out with some rules of thumb, not clearly justified and in some cases based only on their previous experience. In this sense, one of the aims of this work is to analyze differences between different strategies to select the number of neurons in a neural architecture.

From the existing research about the influence of the architecture on the generalization ability there is a kind of agreement in the field [17] that the number of optimal nodes in a neural architecture for obtaining valid generalization depends mainly on three factors: the number of available training samples, the dimension of the function and the complexity of the function. The complexity of the function characterizes how smooth is a given function, and it is clear that the smoother the function the less number of nodes needed and the less number of training samples needed to specify it. The dimensionality of a function might also play an important role given the problem of the curse of dimensionality (See [17]).

It should be also mentioned that there exist alternative approaches for solving the architecture selection problem, like constructive algorithms [14,25,33,35] or pruning techniques [17,19,20]. Constructive methods, in general, generate both the architecture and the set of synaptic weights (i.e., the training of the examples is included in the construction process) starting from a small network and adding neurons as the problem requires it. Many different constructive algorithms have been proposed but they have not been extensively applied in practice, and a reason might be that the obtained generalization ability is lower than the obtained with the traditional back-propagation training approach [33]. Some works have also suggested that large neural networks with small weights trained by back-propagation do not overfit [18] and thus the use of large architectures where the size of the weights is controlled by some regularization technique, like weight decay or early stopping is an alternative. Furthermore, the opposite approach to constructive techniques can also be applied, in this case starting with a large network and after the training process has taken place, a pruning method is applied eliminating less useful weights and neurons [17,19,20].

In this work, by using a recently introduced measure for the complexity of Boolean functions, named generalization complexity (GC measure) [9,11], a new method for selecting neural architectures is proposed and compared to some existing rules. The method can be, in principle, only applied to Boolean input data, but we note that through a pre-processing stage real input data can be transformed into binary data and the method applied to them. Extensive numerical simulations with a large set of well known Boolean functions were carried out, testing different architecture sizes and learning parameters, in order to analyze the role of function complexity regarding the optimal number of hidden neurons in an architecture that leads to the largest generalization ability. The paper is structured as follows: sect. 2 contains the details of the methods utilized in the present study, followed by the results obtained from the extensive numerical simulations that are presented in sect. 3, to finally discuss the results in sect. 4.

2 Methods

2.1 Estimation of Architecture Size by Using the GC Measure

The new method introduced in this work for choosing a neural architecture is based on the use of a complexity measure for Boolean functions, introduced by Franco and colleagues [9–11] and named generalization complexity (GC). The GC-measure, $\mathcal{C}[f]$, was derived from the evidence that bordering examples play an important role on the generalization ability [12,13] and can be computed by counting the number of pairs of neighboring examples at Hamming distances 1 and 2 having opposite outputs. The first term of the complexity measure, $C_1[f]$, has been shown to be related to the average sensitivity of a Boolean function, to the number of examples needed to specify a linearly separable function, and to the query complexity of monotone Boolean functions [11]. The complexity measure terms can also be linked to Harmonic analysis, proven to be extremely useful in a number of areas such as learning theory and circuit complexity [21].

The GC-measure 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. Explicitly, the first term can be written as:

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

where the first factor, $\frac{1}{N_{\text{ex}} * N_{\text{neigh}}}$, 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 neighbor examples at a Hamming distance of 1. The second term $\mathcal{C}_2[f]$ is constructed in an analogous way. The value of the complexity measure using Eq. 1 ranges from 0 to 1.5.

The procedure followed to apply the GC measure for selecting the number of hidden units consists in adjusting the relationship between the optimal number of hidden neurons in a single layer neural network to the generalization complexity of the function, and then use this fit for predicting the number of units in the architecture according to the value of GC for the function under analysis. Clearly the fit, and thus the proposed method, depend on the

set of functions used to construct it, but it has to be said that the aim of the present work is to show that the present approach is feasible and that function complexity can be easily estimated and efficiently used for the task of selecting a neural architecture.

2.2 Some Existing Rules for Selecting the Number of Hidden Units

This section describes four approaches that are currently used in the field of neural networks to choose an architecture. Some of the methods have a theoretical formulation behind, but others are just justified based on experience. The following notation is used: N is the input dimension of the data, N_h represents the number of hidden neurons in the single hidden layer used in the neural architectures and M is used for the output dimension (taken as 1 for all the problems considered). T is used to indicate the number of available training vectors. The four approaches described below correspond to two widely used software frameworks Weka [37] and Neuralware [28] and for two other methods proposed by two researchers [23,3].

1. **Weka** [37] is a popular collection of machine learning algorithms for solving real-world data mining problems. Weka permits to compare the predictive accuracy of different learning algorithms. In the implementation of neural network architectures, Weka estimates by default the architecture size as $(N + M)/2$. It is not clear the justification for the introduction of this rule that has been also suggested on other works [29], based on experimental results, and thus it seems that Weka suggests it based on its simplicity and on previous experience.
2. **Neuralware** [28] is a well known commercial software for the implementation of artificial neural networks. They suggest an architecture with $N_h = T/(5 * (M + N))$ neurons. The rationale behind this rule of thumb is to choose the number of hidden units in a way that there are five training examples for each synaptic weight in the architecture.
3. An important contribution about the approximation capabilities of feed-forward networks was made by **Barron** [3], who pointed out that for some classes of smooth functions a number of nodes equal to $N_h \sim C_f(T/(N \log T))^{1/2}$ optimizes the mean integrated squared error. C_f is the first absolute moment of the Fourier magnitude distribution of f but computing it is not straightforward [2,3]. Thus, in this work C_f was taken to be equal to 1 and thus the estimated number of hidden units used was $N_h = (T/(N \log T))^{1/2}$.
4. **Masters** [23] have suggested to use $N_h = \sqrt{N * M}$ neurons in the hidden layer. This rule comes from results indicating that pyramidal neural structures (in which the number of neurons from layer to layer gets reduced) have shown good generalization ability properties.

2.3 The Set of Test Functions

The set of single output Boolean functions used for the numerical simulations corresponds to the different outputs of a set of 9 multi-output functions. We consider the cases of $N = 10, 11, 12, 13$ input dimensions and for each case the number of single output functions was 59, 70, 97 and 70 respectively. These different numbers of functions arise because the output of the functions considered depends on the size of the input. For example, for the function sum considered with 10 inputs, the possible outputs are the 6 significant bits that can be obtained from adding two numbers of 5 bits length. When this same sum function is analyzed in the case N=11, the number of outputs increases to 7 as this is the number of significant outputs that can be obtained from adding two numbers of length 5 and 6.

For the case of $N = 13$ the number of single output functions that can be obtained is larger than the 70 functions considered but due to computational time restrictions the number of

Table 1 Description of the 70 Boolean functions of $N = 11$ inputs used for the analysis of different strategies to select the number of hidden units in a feed-forward neural network

id.	Function description
1–10	i-or-more clumps. The output of function i is 1 if there are i contiguous input bits ON
11–17	The 7 output bits are the result of the sum of two input numbers of length 6 and 5
18–28	The 11 output bits are the result of the product of two numbers of length 6 and 5
29–33	The 5 output bits are the result of the quotient between two input numbers of length 6 and 5
34–39	The 5 output bits are the result of subtracting a 5 bits number to a 6 bits ones
40–48	The 9 output bits are the result of left-shifting the input number i positions
49–68	The 20 output bits are the result of the square of the input number
69	The majority function of the input bits
70	The comparison function of two input numbers of length 6 and 5

functions analyzed was limited to the same set used for the case of $N = 11$ containing 70 functions.

Six out of the nine functions are arithmetic ones forming part of the set of functions used in microprocessors: the functions sum, product, division, subtraction, shifting and square functions. These functions have been also widely analyzed within the area of threshold circuits [31]. The other three functions used for the tests are Boolean functions previously used in several works and include the i-or-more-clumps, the comparison function and the majority function [9,14,24]. The i-or-more-clumps function is a Boolean function that outputs 1 if there are at least i contiguous bit equals to 1 in the input [14]. In Table 1 the set of functions is described, where the name of the function, a short description, and the number of output functions for the case $N = 11$ are given. For dimensions $N = 10$, 12 and 13, the number of single output functions varies a bit with respect to the $N = 11$ case described in detail in Table 1 as the number of outputs changes with the input dimension. As the input dimension grows, new output bits can be considered for each of the 9 functions indicated in Table 1 (conversely for lower dimensions).

2.4 Simulations

We performed numerical simulations over the set of 59, 70, 97 and 70 Boolean functions of 10, 11, 12 and 13 inputs respectively described before and shown in Table 1 for the case of $N = 11$. The neural architectures used for the simulations include a single hidden layer with a number of neurons between 2 and 30. The use of a single hidden layer neural network is justified on the basis that these architectures can compute any Boolean function and that also this type of network are a standard choice in the field. For each neural network size and set of parameters, a tenfold complete cross validation scheme was used with the whole set of examples that for the case of Boolean functions comprises 2^N examples. Each tenfold cross validation was set up choosing iteratively a test fold, then a random validation

fold, and the remaining eight folds were used for training. The cross validation process was repeated 10 times with a different random seed value. Extreme error values (the minimum and maximum error estimates) were excluded in the computation of the mean generalization ability to reduce the influence of the appearance of local minima in the learning process [32]. In order to avoid overfitting, an early stopping procedure was implemented, in which the generalization error is monitored on the validation set, and the generalization value on a different test set is measured at the minimum of the validation error.

The simulations were run on Matlab code under the Linux operating system on a cluster of 25 Pentium IV 2.0 Ghz PCs interconnected through Openmosix. The learning algorithm used was the scaled conjugate gradient back propagation algorithm [27], that combines the model-trust region approach (used in the Levenberg-Marquardt algorithm) with the conjugated gradient approach. This algorithm have shown superlinear convergence properties on most problems and works well with the early stopping procedure applied to avoid overfitting. A brief description of the parameters and their values used for the simulations carried out using the neural network toolbox in Matlab [7] are given below:

- Starting learning rate: 0.001. Regulate the size of the synaptic weights change (the rate is later adjusted automatically during the training process).
- Maximum number of epochs for training: $5,000$. Stop the training after this maximum number of iterations.
- Minimum performance gradient: $1e-6$. Stop the training if the magnitude of the gradient drops below this value.
- Maximum validation failures: 500. Parameter related to the early stopping procedure.
- Sigma parameter: $5.0e-5$. Determines the change in weight for the calculation of the approximate Hessian matrix in the scaled conjugate gradient algorithm.
- Lambda: $5.0e-7$. Parameter for regulating the indefiniteness of the Hessian.
- Neuron transfer function: sigmoid.
- Weight values initialization: randomly inside the range [0, 0.25]

2.5 Statistical Procedures

In order to compare the results obtained from the different rules analyzed in this work, we have applied a series of statistical tests that were used to measure the statistical significance of the differences observed. We have reported in this work differences in performances only when they were statistically significant.

The measure used to evaluate the performance of a neural network architecture for a given function was the classification accuracy, computed for the test set by averaging the results over different initial conditions. The classification accuracy is just the number of correctly classified examples divided by the total number of examples.

In order to evaluate more precisely the performance of the different neural networks architectures used for each function, Demsar [8] proposes to use the Friedman test [15] on the averaged results when tenfold cross-validation is used as sampling method. The Friedman test is a nonparametric test (similar to ANOVA parametric test) that compares the average ranks of K algorithms ($K > 2$). Under the null hypothesis, the Friedman test states that all the K algorithms perform equivalent and the observed difference is merely random. A p-value of 0.001 was used to consider that the results obtained with a given architecture were better than those obtained with a different one. This relatively low p-value was used in order to minimize the probability of selecting the architectures by purely random fluctuations, and lower p-values were not chosen in order to allow for changes in the size of the neural architectures tested. If the null-hypothesis is rejected (means are significantly different from

each other), the Dunn-Sidak post-hoc test [16] is used to study exactly which means differ from each other. This statistical procedure for multiple comparisons provides an upper bound on the probability that any comparison will be incorrectly found significant. Friedman and Dunn-Sidak tests provide the neural network architecture that produces best generalization accuracy for each function. When several architectures with the same generalization accuracy are selected, the criterium of Occam's razor is applied to choose the network with the smallest number of neurons. For the comparison of results between two different methods, the Wilcoxon signed rank test for zero median was used [36].

3 Results

The procedure to apply the GC-based method to estimate the optimal neural network architecture for a particular function consists first in finding a fit between the adequate architecture size and the GC of a set of test functions. Once the fit has been obtained for a given dimension, then it is only needed to compute the GC of a new function to obtain the estimated number of neurons in the hidden layer from the fitted curve. In our case, the test set were the 70 functions indicated in Table 1 for the case $N = 11$ and 59, 97 and 70 functions for dimensions 10, 12 and 13, respectively. This initial process led us to obtain the optimal neural network architecture for each function (Table 2, column 3) after exploring sizes from 2 to 30 hidden units, and testing statistical significance in difference on generalization accuracy by using the Friedman test on the results from the tenfold cross-validation procedure. A two-way analysis was applied as follows: the model that produced the best mean generalization ability for each function f_i was selected as the control model, and then, after applying the multiple comparison procedure, the simplest model (in number of hidden neurons) that was not significantly different from the control model was selected. For example, in the case of the function with $GC = 0.599$ shown in Table 2, the best performance (averaged over the tenfold cross-validation procedure) was obtained by the model with 17 hidden neurons (control group), and the Friedman test indicated (p-value < 0.001) that there was statistically significant difference in generalization accuracy across architectures sizes. In this case, the multiple comparison Dunn-Sidak post-hoc test indicated (at the 0.05 significance level) that the generalization ability for the architecture with 10 hidden neurons was not statistically different from the control group, and then the value $N_h = 10$ is reported in the table.

In Fig. 1 the results obtained are shown where the optimal architecture size is depicted vs. the GC values for the whole set of 70 functions (the values shown in the figure correspond to columns 1 and 2 in Table 2). Figure 1 shows how the GC measure is correlated with the number of hidden neurons in the best architectures found and does not indicate whether using the GC measure fit is close to optimality or not because what it is important is the generalization ability obtained from the selected architectures, fact that is analyzed below. The figure shows the individual values and also the best fit obtained by using an exponential function with its corresponding confidence intervals. The exponential function leaded to the best fit in terms of the quadratic error among other functions considered like polynomial or logarithmic ones. The fitting procedure was implemented using the Matlab fitting curve procedure and it was not computationally expensive in comparison to the times involved in training the neural networks. For applying the architecture selection process based on the GC measure, a similar fit was used but with only 69 functions instead of the 70, in order to exclude the function under analysis for the architecture selection process. As it is possible to appreciate from the figure, the optimal network size is larger for more complex functions,

Table 2 Generalization ability and number of hidden neurons obtained for the whole set of 70 Boolean functions with $N = 11$ inputs using the new GC method and other 4 existing rules

Complexity	Optimum		GCmethod		Barron	Weka	Masters	Neuralware
	N_h	Gen.	N_h	Gen.	$N_h = 4$	$N_h = 5$	$N_h = 3$	$N_h = 27$
0.002	2	1.000	3	1.000	1.000	1.000	1.000	1.000
0.005	24	0.999	3	0.999	0.999	0.999	0.999	0.999
0.013	2	0.997	3	0.997	0.996	0.996	0.997	0.996
0.031	2	0.993	3	0.992	0.991	0.992	0.992	0.992
0.043	2	0.999	3	0.998	0.997	0.995	0.999	0.996
0.065	11	0.991	4	0.989	0.989	0.986	0.987	0.990
0.097	2	0.993	4	0.996	0.996	0.988	0.996	0.991
0.132	8	0.990	4	0.986	0.986	0.984	0.979	0.985
0.191	8	0.996	4	0.992	0.992	0.984	0.988	0.984
0.225	2	0.990	5	0.979	0.989	0.979	0.992	0.973
0.234	2	0.997	5	0.982	0.991	0.982	0.996	0.975
0.241	7	0.987	5	0.983	0.967	0.983	0.943	0.983
0.244	2	0.984	5	0.972	0.982	0.972	0.984	0.959
0.248	7	0.985	5	0.972	0.982	0.972	0.961	0.968
0.264	2	1.000	5	0.984	0.984	0.984	0.994	0.962
0.264	2	0.997	5	0.976	0.979	0.976	0.995	0.966
0.273	2	1.000	5	1.000	1.000	1.000	1.000	1.000
0.273	2	1.000	5	1.000	1.000	1.000	1.000	1.000
0.348	8	0.989	5	0.974	0.972	0.974	0.954	0.975
0.349	2	0.989	5	0.990	0.968	0.990	0.996	0.963
0.349	2	0.999	5	0.976	0.986	0.976	0.997	0.988
0.373	5	0.993	6	0.977	0.993	0.972	0.982	0.973
0.386	2	0.987	6	0.983	0.975	0.977	0.987	0.984
0.407	7	0.987	6	0.987	0.970	0.965	0.922	0.973
0.419	5	0.983	6	0.988	0.983	0.979	0.970	0.975
0.436	4	1.000	6	0.999	1.000	0.999	0.987	0.948
0.443	2	0.963	6	0.987	0.977	0.979	0.994	0.970
0.452	2	1.000	6	1.000	1.000	1.000	1.000	1.000
0.491	8	0.989	7	0.989	0.964	0.981	0.964	0.993
0.491	4	1.000	7	0.999	1.000	1.000	0.994	0.941
0.492	2	1.000	7	1.000	1.000	1.000	1.000	1.000
0.494	7	0.983	7	0.983	0.981	0.972	0.928	0.971
0.494	8	0.999	7	0.999	0.968	0.991	0.887	0.997
0.495	5	0.981	7	0.996	0.981	0.986	0.937	0.986
0.503	6	0.983	7	0.973	0.978	0.983	0.953	0.976
0.508	7	0.957	7	0.952	0.962	0.956	0.899	0.947
0.509	3	0.990	7	1.000	1.000	1.000	0.990	1.000
0.539	4	0.993	7	1.000	0.999	1.000	0.973	0.963
0.539	7	0.989	7	0.981	0.971	0.974	0.943	0.985

Table 2 continued

Complexity	Optimum		GCmethod		Barron	Weka	Masters	Neuralware
	N_h	Gen.	N_h	Gen.	$N_h = 4$	$N_h = 5$	$N_h = 3$	$N_h = 27$
0.551	5	0.995	7	1.000	0.995	1.000	0.937	0.991
0.562	5	0.992	7	0.995	0.992	0.995	0.931	0.966
0.586	9	0.997	8	0.997	0.991	0.993	0.937	0.984
0.586	7	0.985	8	0.986	0.986	0.981	0.914	0.984
0.599	10	0.976	8	0.974	0.952	0.962	0.881	0.973
0.600	8	0.995	8	0.996	0.987	0.992	0.895	0.995
0.604	5	0.996	8	0.999	0.996	0.996	0.935	0.984
0.607	12	0.980	8	0.957	0.916	0.934	0.896	0.965
0.643	15	0.995	8	0.993	0.962	0.984	0.853	0.998
0.643	6	0.997	8	1.000	0.992	0.997	0.854	1.000
0.700	5	0.996	9	1.000	0.996	1.000	0.942	1.000
0.706	10	0.916	9	0.916	0.830	0.862	0.760	0.928
0.727	14	0.963	9	0.941	0.875	0.927	0.785	0.961
0.734	14	0.993	9	0.989	0.900	0.950	0.750	0.996
0.777	29	0.944	10	0.927	0.799	0.842	0.720	0.941
0.777	4	0.994	10	1.000	0.997	1.000	0.921	1.000
0.798	16	0.978	10	0.962	0.795	0.834	0.688	0.988
0.807	4	0.983	10	1.000	0.994	1.000	0.895	1.000
0.817	6	0.998	11	1.000	0.995	0.998	0.877	1.000
0.818	14	0.983	11	0.978	0.807	0.853	0.678	0.992
0.840	14	0.791	11	0.774	0.684	0.703	0.641	0.818
0.869	16	0.951	11	0.916	0.699	0.751	0.624	0.975
0.871	15	0.864	11	0.849	0.698	0.738	0.644	0.872
0.900	17	0.732	12	0.709	0.636	0.640	0.603	0.748
0.911	14	0.624	12	0.627	0.610	0.623	0.588	0.585
0.912	17	0.818	12	0.724	0.619	0.643	0.576	0.860
0.930	17	0.808	12	0.732	0.614	0.640	0.564	0.856
0.945	6	0.592	13	0.567	0.590	0.592	0.563	0.549
0.956	8	0.577	13	0.583	0.570	0.574	0.563	0.553
0.967	4	0.544	13	0.529	0.536	0.543	0.540	0.527
0.980	7	0.538	13	0.514	0.529	0.536	0.523	0.511
0.516	7.46	0.945	7.40	0.938	0.914	0.9226	0.879	0.939

The column indicated by "Optimum" includes the best results found in the exhaustive numerical simulations with a number of hidden units from 2 to 30. Average values are shown at the bottom row

confirming the fact that the complexity of the function is an important factor in order to select an adequate network size.

Thus, to obtain the optimal neural network size using the new method, the value of the GC of the function is computed and the number of units is obtained from the fitted function previously obtained. The results of the method for the 70 functions are shown in columns 4 and 5 in Table 2, showing both the architecture size estimated by the fitting method and its corresponding generalization accuracy. The results of the other four methods used are also

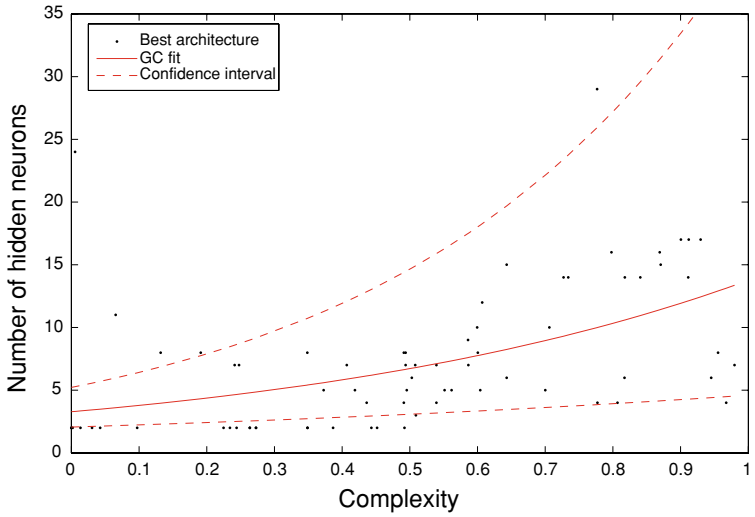

Fig. 1 Optimal architecture size (number of neurons in the single hidden layer) as a function of the Generalization complexity measure (GC) for 70 Boolean functions of 11 input variables. An exponential fitting curve is also shown with the corresponding confidence intervals

shown in the table in columns 6–9. For this four methods, the number of hidden units used are indicated in the second row of the table as they are constant for all functions of the same input dimension.

The results of all 6 different approaches are shown in Fig. 2 showing the generalization ability obtained as a function of the GC measure of the functions. The graphs shown are polynomial fits of the individual values for each of the 70 functions used. The different curves are quite similar for low complexity functions of up to approximately 0.6 (except for the Master rule for which the generalization ability starts to decrease at around 0.4). At this point, the generalization ability of the Barron method and the Weka rule start to go down. The generalization ability remains high for the Neuralware method, for the GC based method and of course for the optimal architectures obtained from exhaustive simulations, but up to around 0.8, where all the curves start to decrease towards 0.5 for functions with a complexity between 0.8 and 1. Note that in the figure, the results of the generalization ability for the Neuralware rule are slightly above the Best architecture ones, fact that in principle cannot be possible but that occurs because we choose the best architecture as the smaller one that shows statistically different results with other size architectures. Thus, the Neuralware curve is not in a statistically significant way above the best architecture curve.

A comparison of the performance of the new GC method against the 4 other analyzed rules is shown in Fig. 3. In this figure, the functions with $N = 11$ were grouped according to their GC complexity in 5 groups and the values displayed are for the cases in which the difference in generalization ability was statistical significantly different, measured using a Wilcoxon signed rank test with $p < 0.05$ [36]. There are less than four bars for each complexity level in the Fig. 3 as the absence of the respective bar indicates that the difference in respect to the use of the GC method was not statistically significant.

The largest differences, in which the GC method outperforms the architectures designed using the Masters, Weka and Barron rules, were found in the middle-high and highest

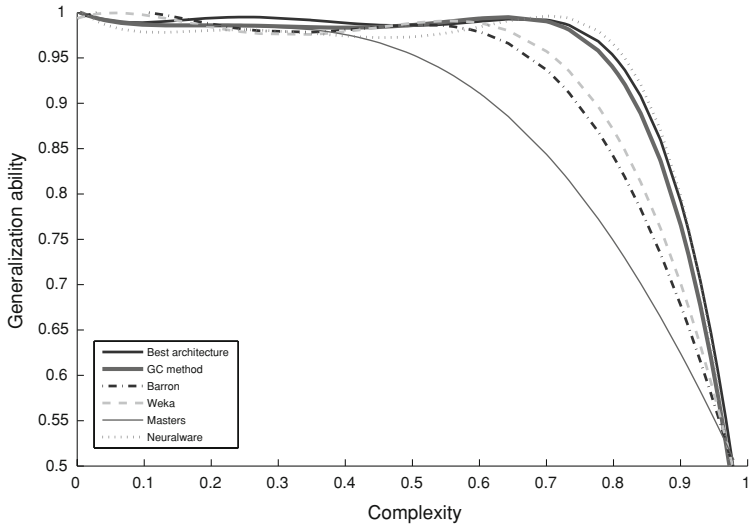

Fig. 2 Generalization ability vs function complexity (GC) for 70 Boolean functions. The *curves* are smoothed fits of the results obtained for the optimum architecture found ("Best architecture"), for the new method introduced ("GC method") and for 4 other existing rules in the literature (See the text for more details).

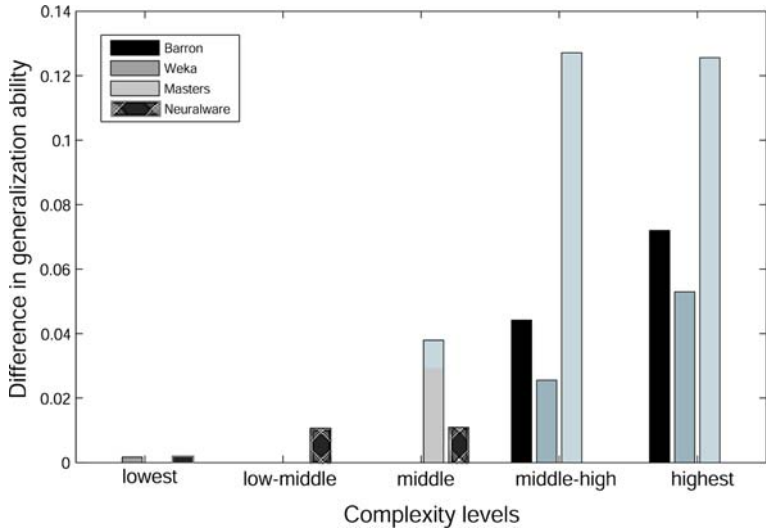

Fig. 3 Difference in generalization ability between the GC based method and the four analyzed rules for Boolean functions on 11 variables grouped according to their complexity. For each level of complexity only the statistically significant differences are reported (See text for more details)

complexity groups. The difference between the GC method and the Neuralware approach was also significant, but very small for the lowest, low-middle and middle groups.

In the last row of Table 2, the average values of network size and generalization ability are computed for the introduced GC method and the other rules analyzed. The mean generalization ability was similar for the GC method and the Neuralware (0.938 and 0.939, respectively) showing that the selection of a large architecture can be a good choice independently of the

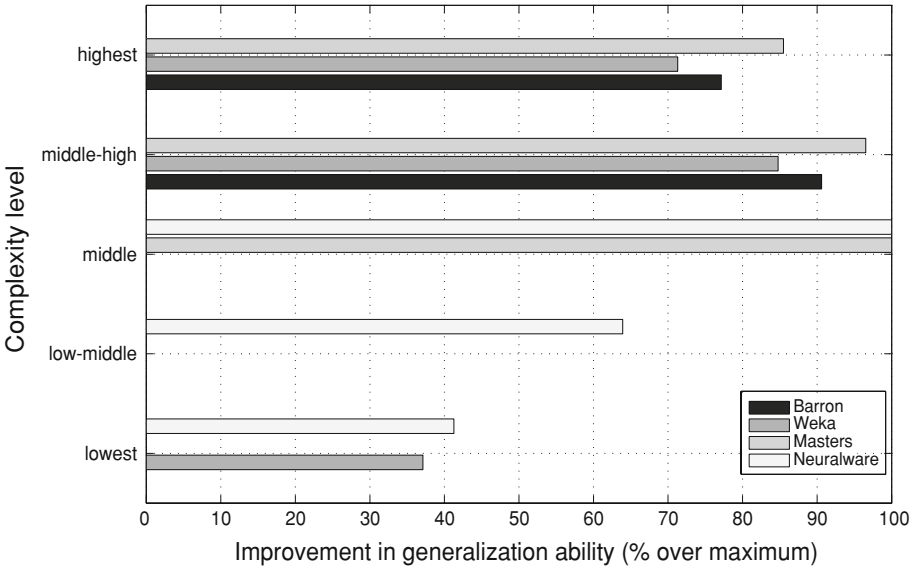

Fig. 4 Difference in generalization ability as a percentage of the maximum that can be obtained between the GC based method and the four rules analyzed for $N = 11$. The absence of a *bar* in each complexity level for any of the 4 methods used for the comparison indicates that the difference was not statistically significant in that case

complexity of the function. Nevertheless, the mean neural network size is lower for the GC method, 7.29 neurons versus the 27 neurons used for the Neuralware rule, and this is a big advantage in the sense of the computational costs involved specially for large dimensional inputs.

We have also computed the difference between the new introduced method and the other 4 analyzed rules, comparing the improvement obtained in relationship to the maximum that can be obtained. Figure 4 shows these values for the case of Boolean functions of 11 variables, while in Fig. 5 the cases for $N = 10, 12$ and 13 are presented. These graphs shows the improvement, I, as a function of the generalization ability obtained from the GC method (G_{gc}) over the other 4 methods as a percentage of the optimum (G_{opt}), computed using the following equation:

$$I = \frac{G_{gc} - G_i}{G_{\text{opt}} - G_i} \times 100.$$

For these experiments, the optimal generalization accuracy is computed by an exhaustive search for the best network topology (columns 2 and 3 in Table 2). Figure 4 shows that, in spite of low values in absolute difference of generalization accuracies, the percentage in improvement exceeds the 50% for several complexity groups of functions, being even up to 100% in some cases, as for the middle groups in dimension 10, 11 and 13, where the GC based method achieved the same level of accuracy than the optimum.

An important issue when analyzing the behavior of the generalization ability is how it scales with the dimension of the problem (number of input variables). Thus, we carried out similar simulations for dimensions 10, 12 and 13. The results are shown in Fig. 5 for the percentages in generalization accuracy improvement of the proposed method over the four other rules, computed in a similar way to the results shown in Fig. 4. In these cases, similar

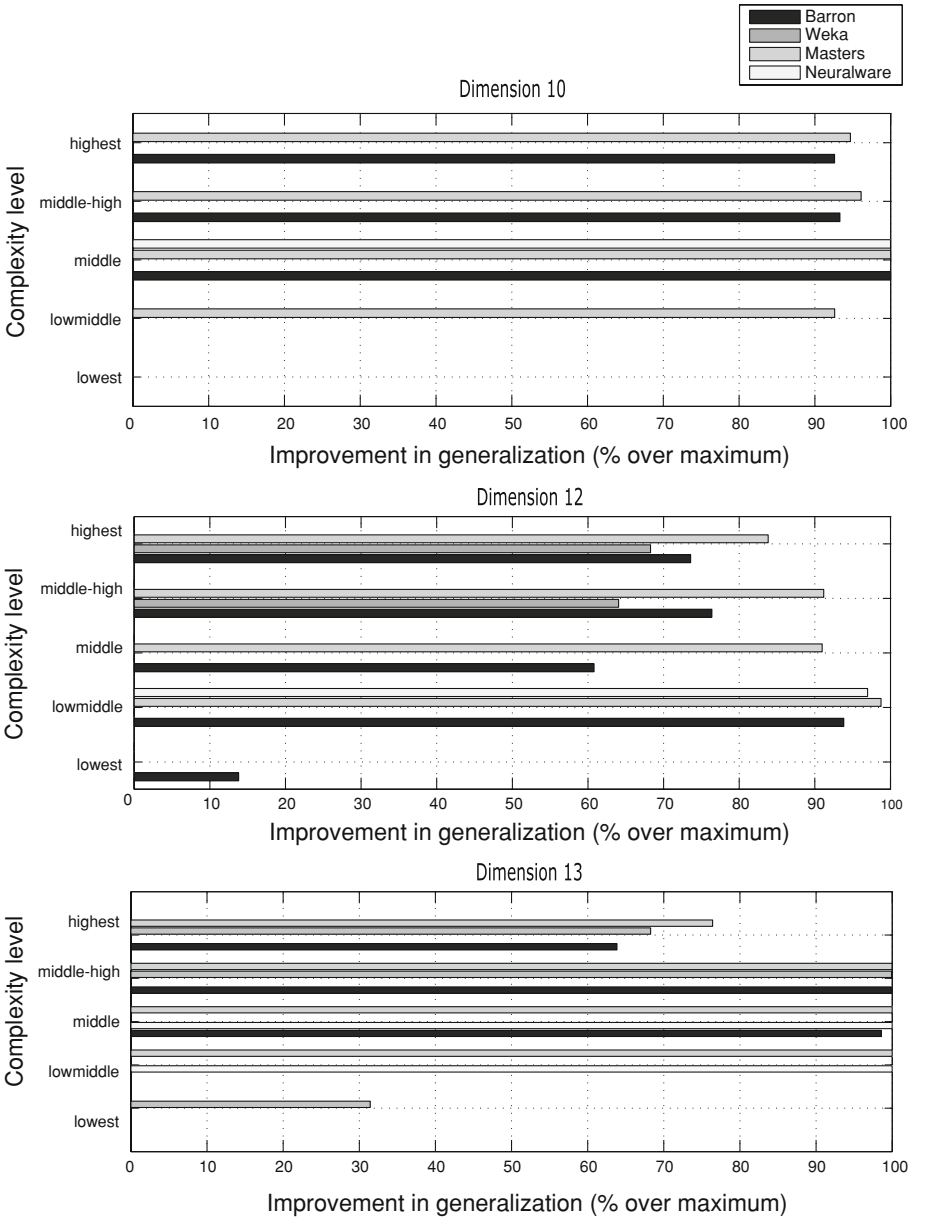

Fig. 5 Difference in generalization ability as a percentage of the maximum that can be obtained between the GC based method and the four rules analyzed for $N = 10$ (*top figure*), $N = 12$ (*middle figure*) and $N = 13$ (*bottom figure*)

results were obtained showing that the GC method can lead to a consistent improvement of the generalization ability for different input dimensions. We also computed the average size of the architectures generated by the GC method for $N = 10$, 11 and 12, obtaining a number of neurons equal to 7.17, 7.64, 8.53 and 9.73, respectively. These values were obtained using the same set of 57 functions common to all 3 sets of functions analyzed.

We note that our analysis was based on 59, 70, 97 and 70 Boolean functions for dimensions $N = 10, 11, 12$ and 13 and these numbers represent only a small fraction of the total number of existing functions. Nevertheless, previous results on lower dimensions, where exhaustive simulations were carried out [9,11], indicate that the relationship between the GC complexity and the generalization ability obtained is on average quite accurate for almost all functions. Also we note that our results were obtained in a leave-one-out procedure, in which the function under analysis is not consider for the fitting procedure involved in the estimation of the number of neurons when the GC method is considered and thus, this fact lead us to conjecture that the procedure should work similarly with a different set of functions. We also note that in the reported results obtained using the Barron's method, we applied a simplified version of the original Barron's equation with a value of C_f equals to 1. As this value leads to small neural architectures, in general poor generalization values were obtained and thus we have also analyzed the case of using a value of C_f equals to 2. For all the set of functions included in Table 2 ($N = 11$ and $N_h = 8$), the average generalization ability obtained was 0.9285, much closer to the results obtained with other methods.

We have also compared the computational times involved in the computation of the GC-measure and for the training of the neural architectures, obtaining that both times scale almost exponentially with the number of inputs (or similarly, scale linearly with the number of training examples), as it can be expected. We have computed the average time needed to compute both mentioned tasks, for functions with a number of inputs between 5 and 14 and neural architectures with a number of neurons between 2 and 20, to obtain that on average training a neural architecture cost 140 times more than computing the value of the GC-measure.

4 Conclusions and Discussion

We have introduced in this work a new approach that permits to select the number of hidden units in a neural network for a given Boolean function. The method is based on a simple measure for the complexity of the functions named generalization complexity (GC). The extensive simulations presented in this work shows that using the GC method for selecting the number of neurons in a feed-forward neural network leads to very efficient results for the generalization ability obtained, in comparison to other four rules tested, and quite close to the optimal values obtained from exhaustive simulations for all possible neural sizes up to 30 neurons. In Fig. 2 we showed the average results obtained for all 70 Boolean functions with 11 inputs, and it can be clearly seen that the new method leads to results that are quite close to the optimum. Figure 3 shows the difference between the GC method and the other four tested rules for the functions grouped according to their complexity value. For low and middle-low complexity functions the difference is not large, but as the complexity increases the GC method outperformed, in a statistical significant way, 3 out of 4 of the other rules tested. From the 4 different rules analyzed, the Neuralware approach was the closest one to the values obtained by the GC method leading to higher generalization ability values for very complex functions, even if this difference was not statistical significant in comparison to the results obtained from the GC method. The difference of the Neuralware approach against the other three ones studied is that this rule leads to architectures with a larger number of neurons that, combined with the early stopping procedure implemented, worked quite well (this result is in agreement with previous published results about the generalization ability of larger networks [18]). It is worth noting that as we select the best architectures on statistical basis, given priority to smaller ones, this procedure has the effect that the fitting curve later

used for the GC-method is biased towards smaller architectures and this explains the fact that using the Neuralware rule leads in some cases to better results for complex functions when larger architectures are needed.

We have also analyzed the difference in generalization ability for different input size functions analyzing the cases for $N = 10,11,12$ and 13 inputs. The obtained results, in terms of the improvement over other alternative rules, were quite similar and show the applicability of the GC method for different input sizes functions (cf. Figs. 4, 5). Nevertheless, a limitation of the new GC-method introduced in this work is that it needs a fitting function that has to be obtained in advance for the dimension of the problem to be studied. On the other hand, when the fit is obtained then only the GC measure has to be computed for a new function in order to estimate the neural architecture, procedure that is much faster than training different size neural architectures. We have done some simulation tests to obtain that computing the GC measure is approximately 140 times faster than training a single neural architecture (more details are given at the end of the results section). We hope that from future studies a general fitting curve for different input sizes can be obtained, permitting the application of the GC method to any input size function. We also note that we have used and analyzed for comparison the results obtained using a simplified version of Barron's formula [2,3], using a raw approximation for the Fourier coefficients needed, as they are quite difficult to compute. We would like to note that, as Barron's formula takes into account the effect of the dimension of the problem considered, it could be combined with other approaches, like the based on the GC measure introduced in this work, to obtain an approximation to the number of hidden neurons needed in neural network architectures for higher dimensions to those considered here.

It is also worth noting that the present method is restricted to Boolean functions as the generalization complexity measure is only defined for this case. However, real value input functions can be discretized using standard available methods and thus it is possible to apply the present approach. We are at the moment analyzing this approach and also working on the extension of the GC measure to real input functions.

Acknowledgments The authors acknowledge support from CICYT (Spain) through grants TIN2005-02984 and TIN2008-04985 (including FEDER funds) and from Junta de Andalucía through grants P06-TIC-01615 and P08-TIC-04026. Leonardo Franco acknowledges support from the Spanish Ministry of Education and Science through a Ramón y Cajal fellowship. We acknowledge useful discussions with Prof. M. Anthony (LSE, London, UK) and with J. L. Subirats (Málaga, Spain).

References

1. Arai M (1993) Bounds on the number of hidden units in binary-valued three-layer neural networks. Neural Netw 6(6):855–860
2. Barron AR (1993) Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans Inform Theory 39(3):930–945
3. Barron AR (1994) Approximation and estimation bounds for artificial neural networks. Mach Learn 14(1):115–133
4. Baum EB, Haussler D (1990) What size net gives valid generalization? Neural Comput 1(1):151–160
5. Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36(4):929–965
6. Camargo LS, Yoneyama T (2001) Specification of training sets and the number of hidden neurons for multilayer perceptrons. Neural Comput 13(12):2673–2680
7. Demuth H , Beale M (1994) MATLAB neural networks toolbox—user's guide version 4. The Math Works, USA
8. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

9. Franco L (2006) Generalization ability of Boolean functions implemented in feedforward neural networks. Neurocomputing 70:351–361
10. 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. pp 973–978
11. Franco L, Anthony M (2006) The influence of oppositely classified examples on the generalization complexity of Boolean functions. IEEE Trans Neural Netw 17(3):578–590
12. Franco L, Cannas SA (2000) Generalization and selection of examples in feedforward neural networks. Neural Comput 12(10):2405–2426
13. Franco L, Cannas SA (2001) Generalization properties of modular networks, implementing the parity function. IEEE Trans Neural Netw 12:1306–1313
14. Frean M (1990) The upstart algorithm, a method for constructing and training feedforward neural networks. Neural Comput 2(2):198–209
15. Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Am Stat Assoc 32:675–701
16. Hájek J, Šidák Z, Sen PK (1999) Theory of rank tests, 2nd ed. Academic Press, Orlando
17. Haykin S (1994) Neural networks (a comprehensive foundation). Prentice Hall, USA
18. Lawrence S, Giles CL, and Tsoi A (1996) What size neural network gives optimal generalization? Convergence properties of backpropagation. Technical Report UMIACS-TR-96-22 and CS-TR-3617, University of Maryland
19. LeCun Y, Denker J, Solla S, Howard RE, Jackel LD (1990) Optimal brain damage. In: Touretzky DSAdvances in neural information processing systems II. Morgan Kaufman, San Mateo
20. Liang X (2006) Removal of hidden neurons in multilayer perceptrons by orthogonal projection and weight crosswise propagation. Neural Comput Appl 16(1):57–68
21. Linial N, Mansour Y, Nisan N (1993) Constant depth circuits, Fourier transform and learnability. J ACM 40:607–620
22. Liu Y, Starzyk JA, Zhu Z (2007) Optimizing number of hidden neurons in neural networks. In: AIAP'07 proceedings of the 25th conference on proceedings of the 25th IASTED international multi-conference. Anaheim, CA, USA, pp 121–126, ACTA Press
23. Masters T (1993) Practical neural network recipes in C++. Academic Press Professional, Inc., San Diego
24. Mayoraz E (1991) On the Power of Networks of Majority Functions'. In: IWANN '91 Proceedings of the international workshop on artificial neural networks. Springer-Verlag, London, UK, pp 78–85
25. Mezard M, Nadal J-P (1989) Learning in feedforward layered networks (the tiling algorithm. J Phys A Math Gen 22(12):2191–2203
26. Mirchandani G, Cao W (1989) On hidden nodes for neural nets. IEEE Trans Circuit Systems 36(5):661–664
27. Moller MF (1993) Scaled conjugate gradient algorithm for fast supervised learning. Neural Netw 6(4):525–533
28. Neuralware I (2001) The reference guide. http://www.neuralware.com
29. Piramuthu S, Shaw M, Gentry J (1994) A classification approach using multi-layered neural networks. Decis Support Syst 11:509–525
30. Scarselli F, Tsoi AC (1998) Universal approximation using feedforward neural networks a survey of some existing methods, and some new results. Neural Netw 11(1):15–37
31. Siu K-Y, Roychowdhury VP (1994) On optimal depth threshold circuits for multiplication and related problems. SIAM J Discret Math 7(2):284–292
32. Vázquez EG, Yanez A, Galindo P, Pizarro J (2001) Repeated measures multiple comparison procedures applied to model selection in neural networks. In: IWANN '01 proceedings of the 6th international work-conference on artificial and natural neural networks. Springer–Verlag, London, UK, pp 88–95
33. Šmieja FJ (1993) Neural network constructive algorithms (trading generalization for learning efficiency?. Circuits Syst Signal Process 12(2):331–374
34. Wang J, Yi Z, Zurada JM, Lu B-L, Yin H (eds) (2006) Advances in neural networks—ISNN 2006, Third international symposium on neural networks, Chengdu, China, May 28–June 1, 2006, Proceedings, Part I, Vol. 3971 of Lecture notes in computer science. Springer
35. Weigend AS, Rumelhart DE, Huberman BA (1990) Generalization by weight-elimination with application to forecasting. In: NIPS. pp 875–882
36. Wilcoxon F (1945) Individual comparisons by ranking methods. Biometrics 1:80–83
37. Witten IH, Frank E (2005) Data mining, practical machine learning tools and techniques. Morgan Kaufmann, San Mateo

38. Yuan HC, Xiong FL, Huai XY (2003) A method for estimating the number of hidden neurons in feed-forward neural networks based on information entropy. Comput Electr Agric 40:57–64
39. Zhang Z, Ma X, Yang Y (2003) Bounds on the number of hidden neurons in three-layer binary neural networks. Neural Netw 16(7):995–1002