Neural Computing and Applications (2020) 32:8955–8963
https://doi.org/10.1007/s00521-019-04388-2

ORIGINAL ARTICLE

Improving learning and generalization capabilities of the C-Mantec constructive neural network algorithm

Iván Gómez[1] · Héctor Mesa[1] · Francisco Ortega-Zamorano[1] · José M. Jerez-Aragonés[1] · Leonardo Franco[1]

Received: 10 May 2018 / Accepted: 24 July 2019 / Published online: 1 August 2019

Abstract

C-Mantec neural network constructive algorithm Ortega (C-Mantec neural network algorithm implementation on MATLAB. https://github.com/IvanGGomez/CmantecPaco, 2015) creates very compact architectures with generalization capabilities similar to feed-forward networks trained by the well-known back-propagation algorithm. Nevertheless, constructive algorithms suffer much from the problem of overfitting, and thus, in this work the learning procedure is first analyzed for networks created by this algorithm with the aim of trying to understand the training dynamics that will permit optimization possibilities. Secondly, several optimization strategies are analyzed for the position of class separating hyperplanes, and the results analyzed on a set of public domain benchmark data sets. The results indicate that with these modifications a small increase in prediction accuracy of C-Mantec can be obtained but in general this was not better when compared to a standard support vector machine, except in some cases when a mixed strategy is used.

Keywords Constructive neural network · Feed-forward network · Support vector machine · C-Mantec · Learning and generalization properties · Loading problem

1 Introduction

Artificial neural networks are biological inspired computational models based in the brain structure and operation, composed of several interconnected nodes or neurons. Some of these units exchange information with the environment (input and output units), and others are internal units communicating only with hidden or internal units within the network. The learning and generalization properties of these models give them a wide applicability in pattern recognition, financial analysis, biology, medicine, image analysis, etc.

Supervised trained artificial neural networks have evolved from the shallow architectures proposed in the 1980s [21] to the well-known deep learning (DL) neural models used nowadays in several applications and that have revolutionized the field of machine learning [12]. Despite the great advances and superior prediction capabilities shown by DL architectures, there are certain aspects that still leave room for other types of neural architectures: (a) DL architectures are very large and need heavy computational resources both in terms of memory and computation, (b) DL has not shown clear improvement over alternative techniques when small data sets are used. Both previous aspects are relevant, in particular, to embedded systems, used in sensors, actuators and in several electronic devices, as small neural architectures with good learning and generalization capabilities are needed for their implementation in embedded devices.

Algorithms that can automatically determine an optimal network architecture according to the complexity of the underlying function embedded in the data set are highly desirable [4, 7]. Efforts toward the network size determination have been made in the literature for many years, and several techniques have been developed, among which we can highlight as more important the decomposition of singular values [26], techniques based on the study of the geometry of classes [1, 7, 14, 28] or the analysis of the entropy of information [27].

✉ Iván Gómez
ivan@lcc.uma.es

[1] E.T.S.I. Informática, Universidad de Málaga, Málaga, Spain

Alternative approaches have been proposed by other authors, among which we can distinguish pruning techniques and constructive methods [2, 5]. The latter option offers the possibility of generating networks that grow as the input data are analyzed. Moreover, the training procedure in the constructive algorithms, considered a computationally expensive problem in the standard feed-forward neural networks, can be done online and relatively fast even in very small electronic devices [17]. Pruning techniques work in the opposite way, as large architectures are used as starting point and neurons and connections are pruned following an optimization criterion. Further, constructive algorithms search for small architecture solutions, offering the possibility of finding networks with minimum size that could match the complexity of the data.

C-Mantec (*Competitive MAjority Network Trained by Error Correction*) is a constructive neural network algorithm [23, 24] that has shown exactly these previously mentioned capabilities, through a competitive scheme that leads to very small one hidden layer neural architectures, and that has been successfully implemented for its use in limited resource hardware such as microcontrollers and embedded systems [17, 18], noting that the use of a small architecture also reduces energy consumption [25].

The C-Mantec algorithm builds neural networks with a single hidden layer of neurons and a single output neuron, adding new nodes to the hidden layer until all training patterns are correctly classified. It combines competition between neurons with a thermal perceptron learning rule [6], adding stability to the acquired knowledge while the architecture of the net grows. At the single-level neuron, the C-Mantec algorithm uses the thermal perceptron rule, while at a global level it performs a competitive strategy between neurons, so obtaining more compact architectures. The main difference with existing constructive algorithms is that at all times, the existing neurons keep learning the information provided by the input patterns without freezing its synaptic weights as it is the standard procedure in most constructive schemes.

This paper presents first in Sect. 2 a study of the training procedure ("loading problem"), carried out in networks trained by C-Mantec and back-propagation algorithms in order to analyze the observed differences, to later analyze in Sect. 3 the improvements implemented in the C-Mantec algorithm with the aim of increasing its generalization performance, by optimizing the hyperplanes of the neurons. Section 3 includes the results obtained on a set of benchmark functions. Finally, the conclusions of the studies carried out are presented in Sect. 4.

2 Methods

2.1 C-Mantec neural network constructive algorithm

We give below some introduction to the way the C-Mantec algorithm functions as it is relevant to understand the rest of results. C-Mantec generates one hidden layer network architectures with a single output neuron. The activation state (S) of each neuron in the hidden layer depends on the N input signals Ψ_i, and on the actual value of the N synaptic weights w_i and a bias b as follows:

$$S = \begin{cases} 1(\text{ON}) & \text{if } \Phi \geq 0 \\ 0(\text{OFF}) & \text{otherwise} \end{cases}$$

where Φ corresponds to the synaptic potential of the neuron and it is defined by:

$$\Phi = \sum w_i \Psi_i - b$$

The thermal perceptron rule modifies the synaptic weights ∇w_i after the presentation of a single input pattern according to the following equation:

$$\nabla w_i = (t - S)\Psi_i T_{\text{fac}}$$

where t is the target value of the presented input and Ψ_i represents the value of input unit i connected to the output by the weight w_i. The T_{fac} factor can be defined as:

$$T_{\text{fac}} = \frac{T}{T_0} e^{-\frac{|\Phi|}{T}}$$

Similar to a simulated annealing process, the value of T decreases with the learning process according to:

$$T = T_0 \left(1 - \frac{I}{I_{\max}}\right),$$

where $I_{\max}$ defines the maximum number of iterations allowed and I is a cycle counter. The thermal perceptron can be seen as a modification of the standard perceptron rule incorporating a modulation factor forcing the neurons to learn only target examples close to the already learned ones. For a deeper analysis of the performance of the thermal rule, see the original paper [6].

Competition between neurons is implemented selecting the neuron with a smallest value of ϕ which will learn the presented input if certain conditions are met (only if the T_{fac} value is larger than the g_{fac} parameter of the algorithm, to prevent the unlearning of the previous stored information), otherwise a new neuron will be added in the network to learn it. The individual temperature T of each neuron is lowered every time its weight is updated. The addition of a new neuron to the network causes a reset of the individual temperature of all neurons to the initial value T_0. The learning process continues until the whole set of input patterns can be classified correctly by the network.

The C-Mantec output neuron computes the majority function (also called the median operator [11]) of the activation of hidden nodes, as previous experiments shown very good capabilities among the set of linearly separable functions ([22]). It outputs a classifier defined by a set of hyperplanes which categorizes new examples. The growing factor parameter determines when to stop the learning cycle, including a new node in the hidden layer.

2.2 Set of benchmark data sets

In order to analyze the performance of the C-Mantec algorithm and the different strategies to improve the accuracy of the classification results, we have selected 15 real input and Boolean output data from the UCI Machine Learning Repository [13] and the DELVE project. The data set selected (name and number of inputs) is indicated in the two first columns of Table 3.

2.3 Analysis of the training procedure ("loading problem") for C-Mantec and back-propagation algorithms

Probably the most interesting aspect of neural networks is its generalization ability. Even if there are some theories explaining these effects, real results for specific architectures and data are complex to be explained by general theories, and thus, there is almost no other alternative than relying in numerical simulations.

Feed-forward networks operate like combinatorial circuits during testing new examples, being this a fast and fully defined phase, but before predictions can be retrieved from a network, they require a training process to load the information into their components. This process of adjusting the synaptic weights so that the mapping system output performs in a proper way has been known as loading problem, being identified in general case as a NP-complete problem in [9, 10]. One way to analyze this training procedure is to train a neural network and then study how each of the neurons present in the hidden layers contributes to the correct performance of the network during the training phase. It has been argued that balanced loading (such as in the back-propagation algorithm case) leads to better generalization ability than unbalanced cases that can occur in constructive algorithms.

In this section, we study how the C-Mantec algorithm generates the load distribution, comparing these results using the back-propagation algorithm, trying to get an insight into how this distribution can influence the generalization capability of the networks.

We computed numerical simulations using the 15 benchmark data sets described in Sect. 2.2, using a size architecture network determined by running the C-Mantec algorithm. As one of the aims of the present study is the possible application of the algorithms in limited resources devices (microcontrollers, embedded systems, etc.) for a useful comparison, the results obtained for the back-propagation algorithm were executed with the same architecture (i.e., one hidden layer architecture containing the same number of neurons obtained for C-Mantec). The back-propagation algorithm was run using the MATLAB neural network toolbox with the default parameter settings using the Levenberg–Marquardt training algorithm. We used a classical data set splitting scheme to present examples to the neural network, in which 75% of the whole set of examples were used for training purposes and the remaining 25% for testing the prediction performance.

In Table 1, the results of experiments for studying the loading distribution are shown, where the first column is a function identifier, the second column represents the method implemented, the 3–8 columns represent the loading information corresponding to every node in the network using the C-Mantec and back-propagation algorithms, and the two last columns indicate the generalization accuracy with the testing set of examples and the standard deviation of the loading information distributed over the nodes. The loading values obtained for each of the neurons present in the single hidden layer of the used architectures are computed as follows: Once the training process of the networks was finished, the fraction of times that the hidden neuron's output coincides with the desired target was analyzed.

Furthermore, we show in Fig. 1 the standard deviation for the loading of the neurons as a function of the generalization ability for back-propagation (right graph) and C-Mantec (left graph), where it can be seen that the average of both quantities is larger for C-Mantec. The results confirm the balanced loading distribution previously observed for the back-propagation algorithm while larger values are obtained for the C-Mantec algorithm. Nevertheless, in terms of the generalization ability observed values are larger for C-Mantec but it is worth noting that for back-propagation the number of neurons in the hidden layer was not optimized but chosen equal to the one selected by the C-Mantec algorithm.

A further analysis was still carried out in order to check and analyze the functioning of the C-Mantec algorithm. In Table 2, the incremental training accuracy is shown for the different analyzed data sets as the architecture grows as needed. The results confirm that the learning process finishes always with the complete training of the patterns, noting that in order to accomplish this, the C-Mantec algorithm removes some patterns considered as noise. In relationship to this issue, in order to avoid overfitting

Table 1 Load distribution information from every separate hyperplane using C-Mantec and back-propagation algorithms

Data	Method	1	2	3	4	5	6	Gen.	STD
Cancer1	CM	0.4733	0.9733					0.9328	0.3535
	BP	0.9194	0.9194					0.9000	0
Card	CM	0.8826	0.8195					0.8304	0.0445
	BP	0.8521	0.8434					0.8434	0.0061
Diabetes	CM	0.6992	0.779	0.7285	0.6562	0.6523		0.7968	0.0530
	BP	0.8007	0.8125	0.7148	0.7617	0.7539		0.7539	0.0391
Ionosphere	CM	0.9529	0.6410					0.8965	0.2205
	BP	0.8448	0.9051					0.9051	0.0426
KsvsKp	CM	0.9713	0.4931					0.9680	0.3381
	BP	0.9765	0.9793					0.9793	0.0019
Sonar	CM	1						0.7391	0
	BP	0.6811						0.6811	0
Bands	CM	0.6721	0.7500	0.4016	0.7172	0.6106		0.7190	0.1381
	BP	0.6942	0.6446	0.6363	0.6694	0.6528		0.6528	0.0229
Fertility	CM	0.8805	0.2985					0.8484	0.4115
	BP	0.7878	0.7575					0.7575	0.0214
Haberman	CM	0.7512	0.7720	0.7461	0.6424	0.2746		0.7291	0.2088
	BP	0.6145	0.7187	0.7083	0.6979	0.6875		0.6875	0.0412
HeartStatlog	CM	0.8777	0.6944					0.8222	0.1296
	BP	0.8444	0.8444					0.8444	0
Heartc	CM	0.8944	0.4888	0.5555				0.8333	0.2174
	BP	0.8555	0.8111	0.8000				0.8000	0.0293
Mammographic	CM	0.7473	0.7952	0.8085	0.6515	0.3909		0.7433	0.1722
	BP	0.7326	0.7326	0.7272	0.7112	0.7058		0.7058	0.0125
Phoneme	CM	0.7240	0.7263	0.7708	0.7358	0.7288	0.7083	0.7840	0.0209
	BP	0.7924	0.7969	0.7891	0.8065	0.8104	0.8222	0.8222	0.0124
Pima	CM	0.7089	0.7500	0.7519	0.7832	0.6503		0.7578	0.0512
	BP	0.7656	0.7812	0.7500	0.7070	0.6796		0.6796	0.0422
Wisconsin	CM	0.4733	0.9666					0.9530	0.3488
	BP	0.9463	0.9463					0.9160	0

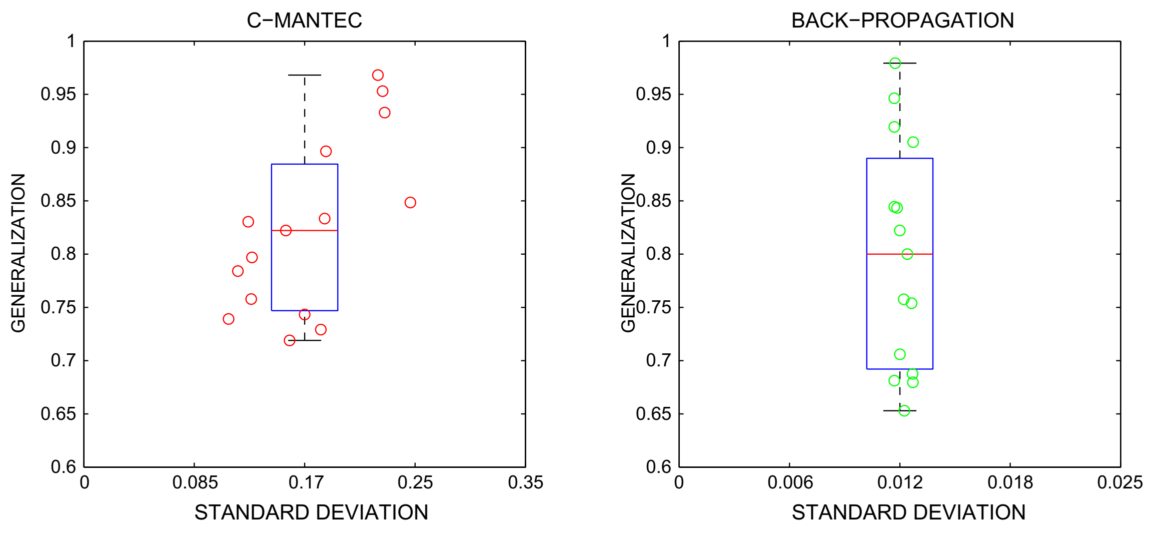
The two last columns shown the generalization results and the standard deviation of the load distribution from C-Mantec method

effects C-Mantec includes built-in process that remove patterns considered as conflictive when they need a certain number of learning modifications (the standard setting consists in eliminating patterns needing more than the average of the rest of the patterns plus two standard deviations). Further details of the procedure can be found in the original C-Mantec papers, where the algorithm was introduced [23, 24].

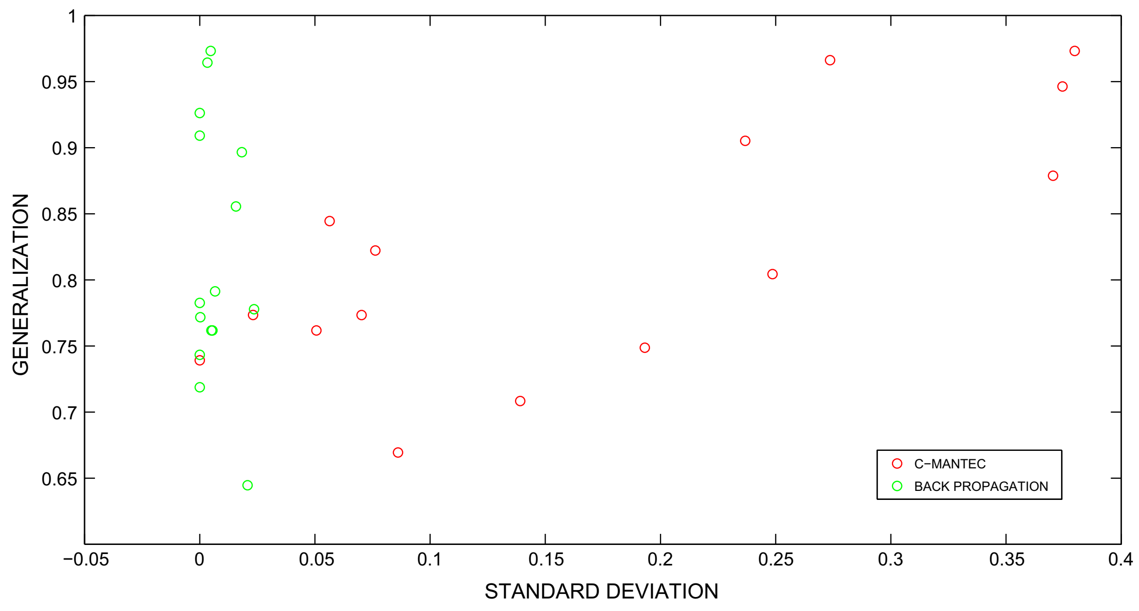
3 Optimizing C-Mantec hyperplane classification

The general observation about the performance of C-Mantec is that it often generates architectures with good prediction capabilities. The addition of new neurons into the hidden layer of the architecture occurs when a new training example cannot be classified correctly by the network in the pre-defined number of iterations. The addition of new nodes to the architecture might allow for new and better hyperplane solutions, and even if the training implemented in C-Mantec permits all neurons to learn, it does not guarantee the optimal option. In order to test how optimal the hyperplane selection was done with the standard C-Mantec settings, we consider in this section different options to optimize the hyperplane position.

A reasonable strategy is to select hyperplanes which have the largest separation to the nearest data point of each class. This situation is well known in the neural networks area, and it is a standard machine learning principle, known as the maximum margin classification [8]. Support vector machines [3] use this principle to classify data, applying a technique called kernel trick to convert a non-separable problem to separable.



(a) Standard deviation vs. generalization for C-Mantec and back-propagation algorithms. The central mark is the median, and the edges of every box are the 25th and 75th percentiles



(b) Standard deviation vs generalization for both C-Mantec and back-propagation algorithms drown on the same scale

Fig. 1 Generalization results vs standard deviation from the loading distribution with C-Mantec and the back-propagation algorithms

According to the above strategy, we present three offline approaches to create the new optimized hyperplane, named maximum margin (MM), parallel to nearest neighbor (PNN) and mixed, that offer the possibility of improving the accuracy of the classifier generated by the C-Mantec algorithm. The three mentioned approaches consider the architecture determined by the application of the C-Mantec method as the starting point, allowing the generation of a new network configuration by modifying slightly every hyperplane of the C-Mantec original solution through certain operations involving the support vectors of every class.

3.1 Maximum margin strategy (MM)

The MM strategy focuses on modifying each C-Mantec classifier individually by an alteration of it, using the n nearest training examples of every outcome half-space. First, the algorithm works with the patterns classified in one class, removing those misclassified and generating a

Table 2 Incremental load distribution obtained from the C-Mantec application in training phase

Data	Nodes					
	1	2	3	4	5	6
Cancer1	0.4733	1				
Card	0.8826	1				
Diabetes	0.6992	0.8016	0.8016	0.8016	1	
Ionosphere	0.9529	1				
KsvsKp	0.9713	1				
Sonar	1					
Bands	0.6721	0.8074	0.8074	0.8607	1	
Fertility	0.8805	1				
Haberman	0.7512	0.7688	0.7688	0.2850	1	
HeartStatlog	0.8777	1				
Heartc	0.8944	0.8000	1			
Mammographic	0.7473	0.9661	0.9831	0.9831	1	
Phoneme	0.7240	0.8551	0.8551	0.8551	0.8591	1
Pima	0.7089	0.8721	0.8721	0.8721	1	
Wisconsin	0.4733	1				

group (G_+) of patterns sorting out by their distance to the original C-Mantec classifier surface. Only these nearest n elements are saved in this group. The same process is done with the opposite class, generating the group G_-.

The algorithm continues selecting the first pattern of G_+ group and matching it with the nearest pattern of the opposite group G_-. Once this correspondence is hold out, the midpoint is computed and the two patterns are dropped from their respective group. Carrying out the same operation with all patterns in G_+, we obtain the n midpoints to compute/build the new MM hyperplane H_{MM}.

We considered not to apply the MM strategy if the number of examples included in any of the classes is lower than n. Figure 2 shows a scheme of the MM strategy implementation.

3.2 Parallel to nearest neighbor strategy (PNN)

The PNN strategy considers the two groups of patterns computed from the MM strategy, and it has two parts: Once the G_+ group is generated, the algorithm computes the hyperplane H_+ defined by the n nearest points of G_+ to the C-Mantec hyperplane H_{CM}. This hyperplane H_+ is considered to find the nearest point x_j belonging to the opposite group, and computing the hyperplane through this point and being parallel to H_+, H'_+. Finally, the bisector hyperplane determined by H_+ and H'_+ is considered as candidate, labeled as H_+*. Following the same technique, the algorithm considers the opposite group of patterns, G_-, to determine H_-*. Figure 3 shows a scheme of the PNN strategy implementation.

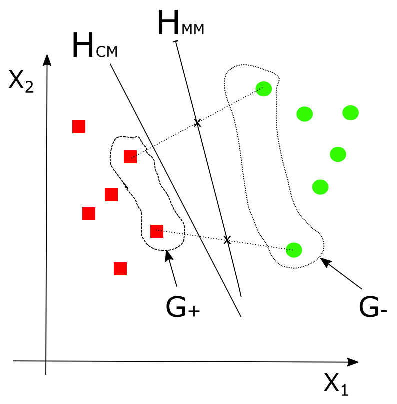


Fig. 2 Maximum margin strategy (MM). H_{CM} is the actual C-Mantec hyperplane. The G_+ class is filled with closest examples to the C-Mantec hyperplane, x_i and x_j. G_- is generated following the same procedure. H_{MM} is computed with the midpoints between pairs of n closest point at different sides of the hyperplane

3.3 Mixed strategy

The Mixed strategy starts with a performance of the C-Mantec algorithm with a standard set of parameters. Thus, considering each hyperplane generated from the C-Mantec method individually, the MM and PNN

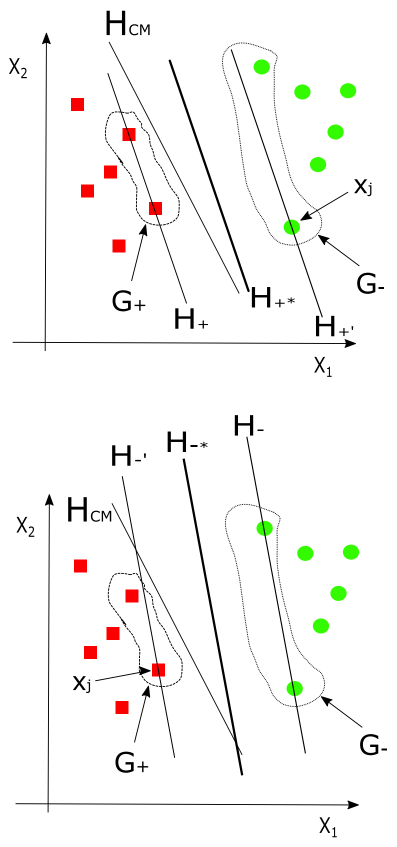


Fig. 3 PNN strategy scheme. H_+* and H_-* are the hyperplane candidates

strategies are computed, and the results are compared to each other. The option with the best accuracy values using the training set will be selected to replace the C-Mantec hyperplane. We note that the application of the Mixed method could provide with architectures without changes in the C-Mantec architecture when the computation of the new strategies in each hyperplane does not improve generalization values.

3.4 Simulations and results

We have performed exhaustive simulations to find the best set of hyperplanes applying the MM and PNN strategies or a combinations of them, named as Mixed strategy, to modify each hyperplane resulting from the application of C-Mantec algorithm. In order to carry out the experiments, we have considered a data partition scheme with ten random boxes, taking nine of them as training and the remaining one as test. Experiments and analysis have been carried out in MATLAB, 2012b version. For further comparison, simulations were carried out also for a standard support vector machine (SVM) with polynomial kernels run with standard settings under MATLAB.

The data and simulation results are summarized in Table 3. The first two columns indicate the name and the number of input variables of each problem. The third and fourth columns refer to the generalization ability obtained from the application of the standard C-Mantec algorithm together with the final training accuracy, which is always 1 for standard C-Mantec as it learns up to perfect training using an elimination procedure of patterns considered as noise. The six following columns correspond to the results in accuracy with test examples and train patterns using the MM and PNN strategy and also an SVM model used for comparison. The last five columns show the results for the Mixed strategy, where in the last three columns the values correspond to the percentage of hyperplanes used for the Mixed strategy.

A difference between the generalization ability for the test patterns is appreciate in Table 3, where we can see an improvement in these values obtained for the majority of the problems considered when the MM strategy is applied, showing/reflecting the worst results with the performance of the PNN strategy in isolation. As might be expected, the prediction ability concerning with a Mixed strategy produces an interesting improvement in a high proportion of the considered data set.

A more detailed analysis of Table 3 reveals an effective performance of the C-Mantec algorithm for the data with a low and medium complexity (highest generalization values with test patterns). As the complexity of the data grows, we can emphasize that the times when the PNN technique is involved overcome the use of the MP technique. Regarding the comparison with the values obtained from a SVM, we

Table 3 Generalization ability obtained in a set of benchmark data sets for C-Mantec (CM) and three alternative hyperplane optimization solutions (MM, PNN, Mixed)

Data	Inputs	CM		MM		PNN		SVM		Mixed				
		Gen	Train	Gen	Train	Gen	Train	Gen	Train	Gen	Train	%CM	%MM	%PNN
KsvsKp	36	0.9748	1	0.9748	1	0.9748	1	0.9587	0.9587	**0.9748**	1	100	0	0
Cancer1	9	0.9559	1	0.9568	0.9999	0.8283	0.8560	0.9530	0.9533	**0.9604**	0.9972	90	5	5
Wisconsin	10	0.9532	1	0.9572	0.9969	0.8132	0.8478	**0.9700**	0.9329	0.9572	0.9969	90	4	6
Ionosphere	34	**0.8811**	1	0.8811	1	0.8811	1	0.8793	0.9274	0.8811	1	100	0	0
Fertility	10	0.8700	1	0.8780	0.9861	0.8640	0.9787	**0.8788**	0.8806	0.8780	0.9782	75	8	17
Card	24	0.8478	1	0.8478	1	0.8478	1	**0.8609**	0.8652	0.8478	1	100	0	0
Heartc	13	0.8365	1	0.8365	1	0.8365	1	**0.8788**	0.8500	0.8365	1	100	0	0
HearthSt	13	0.8193	1	0.8222	0.9963	0.7956	0.8880	0.7556	0.8722	**0.8407**	0.9737	81	4	15
Mammographic	6	0.7884	1	0.7894	0.9957	0.6208	0.6922	**0.7914**	0.7553	0.7855	0.9819	83	3	14
Pima	8	0.7730	1	0.7779	0.9890	0.7683	0.9494	**0.7852**	0.7715	0.7846	0.9714	70	8	22
Phoneme	5	0.7729	1	0.7723	0.9966	0.7566	0.9485	**0.7751**	0.7754	0.7690	0.9827	55	7	38
Diabetes	20	0.7703	1	0.7787	0.9866	0.7622	0.9347	**0.7813**	0.7832	0.7792	0.9770	61	7	32
Sonar	60	0.7528	1	0.7537	0.9954	0.5303	0.5374	**0.7826**	0.8921	0.7576	0.9906	98	0	2
Haberman	3	0.7440	1	0.7481	0.9890	0.7501	0.9697	0.7292	0.7254	**0.7585**	0.9856	70	2	28
Bands	39	0.6729	1	0.6735	0.9977	0.6735	0.9968	0.6612	0.7295	**0.6735**	0.9983	98	2	0
Average		0.8245	1	0.8265	0.9929	0.7770	0.9074	**0.8294**	0.8848	0.8285	0.9883			

Also, the results of generalization and training with a support vector machine are included. For the mixed strategy, the percentage o hyperplanes corresponding to each of the original strategies is shown

Bold values indicate the best results obtained with any of the tested methods see that the SVM leads to slightly larger prediction values that are almost similar to those that can be obtained using the Mixed strategy.

One interesting aspect of the current comparison done between C-Mantec, back-propagation and SVMs is its possible hardware implementation in microcontrollers and embedded systems like FPGAs. In the previous works [17, 19], we implemented both C-Mantec and the back-propagation algorithm in FPGAs and microcontrollers, and the comparison shows that C-Mantec needs lower hardware resources than back-propagation (considering the same architecture) due to the different transfer functions they use [20]. Regarding the comparison between C-Mantec and SVM, based on published bibliography [15] and our experience the situation looks similar, and even more favorable to C-Mantec given that SVMs usually utilize a much larger number of kernel units than C-Mantec.

4 Conclusions and future work

We have analyzed in this work the functioning of the C-Mantec constructive neural network algorithm, carrying out a deep study of how the training of the patterns is performed, a problem known as the loading problem, considered of interest as it is related to the generalization ability that can be obtained. The analysis done in Sect. 2 shows that C-Mantec has a loading distribution among the neurons in the hidden layer that is more unbalanced than that obtained for the well-known back-propagation algorithm, with a value of the standard deviation of 0.17. A more detailed analysis of every individual case shows that in most cases, the deviation values were lower (below 0.05) but in some cases in which only very few neurons were needed the deviation values were higher, with several cases with values approximately around 0.35, but noting that even in these cases the generalization ability observed was still high. The conclusion of this part of the study might suggest that for complex problems where many neurons are needed the loading distribution is quite balanced as expected because C-Mantec permits all neurons to learn simultaneously. In the second part of this work, we analyzed different alternative optimization strategies for the separating hyperplanes of the hidden neurons for the C-Mantec algorithm, having found that on average only a relatively small increase in performance can be obtained, but that for the case of complex problems (for which we used as indication the generalization obtained) some more benefit can be observed mainly by using the PNN (parallel to nearest neighbor) strategy.

As an overall conclusion, we can say that the standard version of the C-Mantec algorithm performs quite well and that in order to obtain larger accuracies like the ones observed in recent year through the use of DL models, it might be necessary to include further layers rather than thinking on optimizations of the current model, as no big improvements have been found with the analyzed alternatives. Nevertheless, one important point observed before and confirmed in this work is that the overall performance of C-Mantec is very competitive in comparison with alternative machine learning models, such as standard back-propagation and SVM, noting that a big advantage of C-Mantec models is the lower number of units present in the generated models that make them ideal for the implementation in limited resource hardware, as is the case of micro-controllers, embedded systems, etc.

Acknowledgements The authors acknowledge support through Grants TIN2014-58516-C2-1-R and TIN2017-88728-C2 from MINECO-SPAIN and from Universidad de Málaga (Plan propio) which include FEDER funds.

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. Augasta MG, Kathirvalavakumar T (2013) Pruning algorithms of neural networks—a comparative study. Cent Eur J Comput Sci 3(3):105–115
3. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297
4. Franco L (2006) Generalization ability of boolean functions implemented in feedforward neural networks. Neurocomputing 70(1–3):351–361
5. Franco L, Elizondo DA, Jerez J (2009) Constructive neural networks, 1st edn. Springer, Berlin
6. Frean MR (1992) A "thermal" perceptron learning rule. Neural Comput 4(6):946–957
7. Gómez I, Franco L, Jerez JM (2009) Neural network architecture selection: can function complexity help? Neural Process Lett 30(2):71–87
8. Gong Y, Xu W (2007) Machine learning for multimedia content analysis (multimedia systems and applications). Springer, New York
9. Judd S (1987) Learning in networks is hard. In: Proceedings of the first IEEE neural network conference (San Diego), pp. II–685–692
10. Judd S (1988) On the complexity of loading shallow neural networks. J Complex 4(3):177–192
11. Knuth D (2008) Introduction to combinatorial algorithms and boolean functions. Art of computer programming: newly available sections of the classic work/Donald E. Knuth. Addison-Wesley, Boston
12. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
13. Lichman M (2013) UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine, CA. http://archive.ics.uci.edu/ml
14. Mirchandani G, Cao W (1989) On hidden nodes for neural nets. IEEE Trans Circuits Syst 36(5):661–664
15. Orozco-Duque A, Rúa Pérez S, Zuluaga S, Redondo A, Restrepo J, Bustamante J (2013) Support vector machine and artificial neural network implementation in embedded systems for real time arrhythmias detection. In: BIOSIGNALS 2013—

proceedings of the international conference on bio-inspired systems and signal processing. pp 310–313
16. Ortega F (2015) C-mantec neural network algorithm implementation on matlab. https://github.com/IvanGGomez/CmantecPaco. Accessed 10 July 2019
17. Ortega F, Jerez J, Franco L (2014) Fpga implementation of the c-mantec neural network constructive algorithm. IEEE Trans Ind Informatics 10(2):1154–1161
18. Ortega-Zamorano F, Jerez J, Subirats J, Molina I, Franco L (2014) Smart sensor/actuator node reprogramming in changing environments using a neural network model. Eng Appl Artif Intell 30:179–188
19. Ortega-Zamorano F, Jerez JM, Urda D, Luque Baena RM, Franco L (2016) Efficient implementation of the backpropagation algorithm in fpgas and microcontrollers. IEEE Trans Neural Netw Learn Syst 27:1840–1850
20. Ortega-Zamorano F, Jerez JM, Juárez GE, Franco L (2017) Fpga implementation of neurocomputational models: comparison between standard back-propagation and c-mantec constructive algorithm. Neural Process Lett 46:899–914
21. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536
22. Subirats JL, Franco L, Gomez I, Jerez JM (2008) Computational capabilities of feedforward neural networks: the role of the output function. Proc XII CAEPIA 7:231–238
23. Subirats JL, Jerez JM, Gómez I, Franco L (2010) Multiclass pattern recognition extension for the new c-mantec constructive neural network algorithm. Cogn Comput 2(4):285–290
24. Subirats JL, Franco L, Jerez JM (2012) C-mantec: a novel constructive neural network algorithm incorporating competition between neurons. Neural Netw 26:130–140
25. Urda D, Cañete E, Subirats JL, Franco L, Llopis L, Jerez J (2012) Energy-efficient reprogramming in wsn using constructive neural networks. Int J Innov Comput Inf Control 8:7561–7578
26. 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
27. 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 Electron Agric 40:57–64
28. 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