Active Learning Using a Constructive Neural Network Algorithm

José Luis Subirats[1], Leonardo Franco[1], Ignacio Molina Conde[2], and José M. Jerez[1]

[1] Departamento de Lenguajes y Ciencias de la Computación
Escuela Técnica Superior de Ingeniería en Informática
Universidad de Málaga,
Campus de Teatinos S/N, 29071 Málaga, Spain
[2] Departamento de Tecnología Electrónica
Escuela Técnica Superior de Ingeniería en Telecomunicación
Universidad de Málaga,
Campus de Teatinos S/N, 29071 Málaga, Spain

Abstract. Constructive neural network algorithms suffer severely from overfitting noisy datasets as, in general, they learn the set of examples until zero error is achieved. We introduce in this work a method for detect and filter noisy examples using a recently proposed constructive neural network algorithm. The method works by exploiting the fact that noisy examples are harder to be learnt, needing a larger number of synaptic weight modifications than normal examples. Different tests are carried out, both with controlled experiments and real benchmark datasets, showing the effectiveness of the approach.

1 Introduction

A main issue at the time of implementing feed-forward neural networks in classification or prediction problems is the selection of an adequate architecture [1,2,3]. Feed-forward neural networks trained by back-propagation have been widely used in several problems but yet the standard approach for selecting the number of layers and number of hidden units of the neural architecture is the inefficient trial-by-error method. Several constructive methods and pruning techniques [1] have been proposed as an alternative for the architecture selection process but it is a research issue whether these methods can achieve the same level of prediction accuracy. Constructive algorithms start with a very small network, normally comprising a single neuron, and work by adding extra units until some convergence condition is met [4,5,6,7]. On the other hand, pruning techniques start with a very large architecture and work by eliminating unnecessary weights and units [8].

Despite the existence of many different constructive algorithms, they have not been extensively applied in real problems. This fact is relatively surprising, given that they offer a systematic and controlled way of obtaining an architecture and also because they offer the possibility of an easier rule extraction procedure. In a 1993 work, Smieja [9] argued that constructive algorithms might be more

V. Kůrková et al. (Eds.): ICANN 2008, Part II, LNCS 5164, pp. 803–811, 2008.

efficient in terms of the learning process but cannot achieve a generalization ability comparable to back-propagation neural networks. Smieja arguments were a bit speculative more than based on obtained results, but nevertheless might explain the fact that constructive methods have not been widely applied to real problems. In recent years new constructive algorithms have been proposed and analyzed, and the present picture might have changed [7,10].

One of the problems that affects predictive methods in general, is the problem of overfitting [11,12]. In particular, overfitting affects severely neural network constructive algorithms as they, in general, learn towards zero error. The overall strategy in constructive algorithms for avoiding overfitting is by creating very compact architectures. Unfortunately, this approach is not enough when the input data is noisy as it is normally the case of real data. A solution to this overfitting problem might be the implementation of methods that exclude noisy instances from the training dataset [13,14,15,16,17]. In this work, we use a recently introduced constructive neural network algorithm named C-Mantec [18] for detecting noisy examples. The method can detect and filter noisy instances leading to an improvement in the generalization ability of the algorithm and permitting to obtain more compact neural network architectures.

2 The C-Mantec Algorithm

The C-Mantec algorithm is a constructive neural network algorithm that creates architectures with a single layer of hidden nodes with threshold activation functions. For functions with 2 output classes, the constructed networks have a single output neuron computing the majority function of the responses of the hidden nodes (i.e., if more than half of the hidden neurons are activated the output neuron will be active). The learning procedure starts with an architecture comprising a single neuron in the hidden layer and adds more neurons every time the present ones are not able to learn the whole set of training examples. The neurons learn according to the thermal perceptron learning rule proposed by Frean [5], for which the synaptic weights are modified according to Eq. 1.

$$\delta w_i = (t - o)\ \psi_i\ \frac{T}{T_0}\ \exp\{-\frac{|\phi|}{T}\}\ , \tag{1}$$

where t is the target value of the example being considered, o represent the actual output of the neuron and ψ is the value of the input unit i. T is an introduced temperature, T_0 the starting temperature value and ϕ is a measure of how far is the presented example from the actual synaptic vector. The thermal perceptron is a modification of the perceptron rule that incorporates a modulation factor forcing the neurons to learn only target examples close to the already learnt ones, in order to avoid forgetting the stored knowledge. For a deeper analysis of the thermal perceptron rule, see the original paper [5].

At the single neuron level the C-Mantec algorithm uses the thermal perceptron rule, but at a global level the C-Mantec algorithm incorporates competition between the neurons, that makes the learning procedure more efficient and permitting to obtain more compact architectures [18]. Competition between neurons is

implemented as follows: for a given input example, the neuron with the smallest value of the parameter ϕ is picked as the neuron that will learn the presented input (note that the weights are modified only for wrongly classified inputs). The temperature, T, controlling each individual perceptrons, is lowered every time the neuron gets an update of its weights. When a new unit is added to the network, the temperature of all neurons is reset to the initial value T_0. The learning procedure continues in this way until enough neurons are present in the architecture, and the network is able to learn the whole sets of inputs. Regarding the role of the parameters, an initial high temperature T_0 ensures a certain number of learning iterations and an initial phase of global exploration for the weights values, as for high temperature values changes are easier to be accepted. The parameter setting for the algorithm is relatively simple as C-Mantec has been shown to be very robust to changes. The convergence of the algorithm is ensured because the learning rule is very conservative in their changes, preserving the acquired knowledge of the neurons and given by the fact that new introduced units learn at least one input example. Tests performed with noise-free Boolean functions using the C-Mantec algorithm show that it generates very compact architectures with less number of neurons than existing constructive algorithms [18]. However, when the algorithm was tested on real datasets, it was observed that a larger number of neurons was needed because the algorithm overfit noisy examples. To avoid this overfitting problem the method introduced in the next section is developed in this work.

3 The "Resonance" Effect for Detecting Noisy Examples

In Fig. 1, an schematic drawing shows the "resonance" effect that is produced when a thermal perceptron tries to learn a set of instances containing a

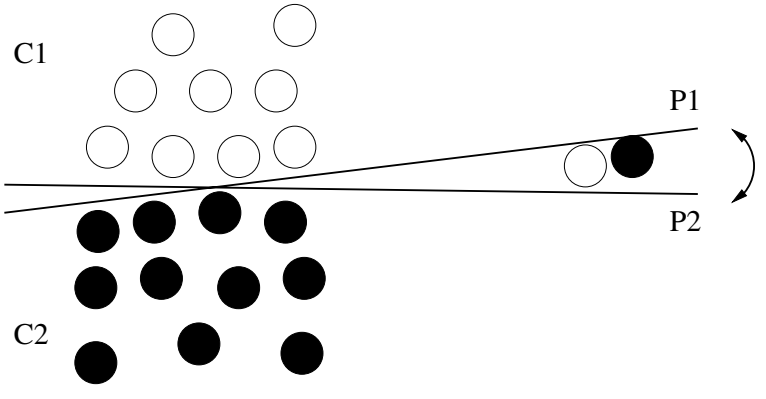

Fig. 1. Schematic drawing of the "resonance effect" that occurs when noisy examples are present in the training set. A thermal perceptron will learn the "good" examples, represented at the left of the figure, but will classify rightly only one of the noisy samples. Further learning iterations in which the neuron tries to learn the wrongly classified example will produce an oscillation of the separating hyperplane. The number of times the synaptic weights are adjusted upon presentation of an example can be used to detect noisy inputs.

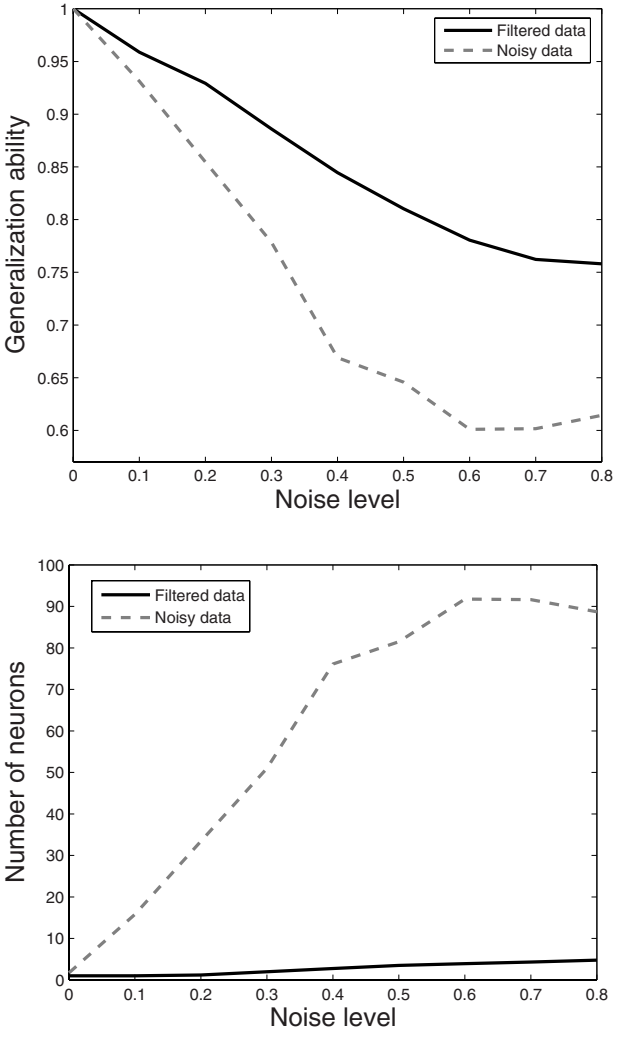

Fig. 2. The effect of attribute noise. Top: Generalization ability as a function of the level of attribute noise for the "modified" Pima indians diabetes dataset for the C-Mantec algorithm applied with and without the filtering stage. Bottom: The number of neurons of the generated architectures as a function of the level of noise. The maximum number of neurons was set to 101.

contradictory pair of examples. In the figure, the set of "good" examples is depicted in the left part of the figure, while the contradictory pair is on the right. When a single neuron tries to learn this set, the algorithm will find an hyperplane from a beam of the possible ones (indicated in the figure) that classifies correctly the whole set except for 1 of the noisy examples. Further learning iterations produce a resonant behavior, as the dividing hyperplane oscillates trying

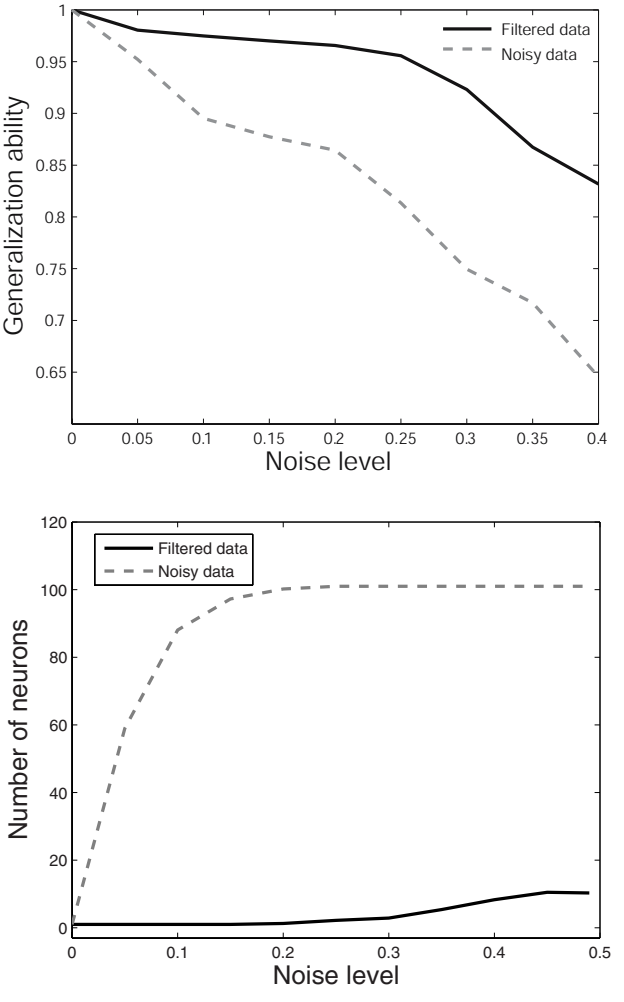

Fig. 3. The effect of class noise. Top: Generalization ability as a function of the level of class noise for the modified Pima indians diabetes dataset for the cases of implementing the filtering stage and for the case of using the whole raw dataset. Bottom: The number of neurons of the generated architectures for the two mentioned cases of the implementation of the C-Mantec algorithm.

to classify correctly the wrong example. Eventually, the iterations will end and as the whole set cannot be learnt, a new neuron will be added to the network. It was observed that these noisy examples make the network to grow excessively and degrade the generalization ability, and thus a method for removing them is quite useful. The method is based on counting the number of times each training example is presented to the network; and if the number of presentations for an example is larger by two standard deviations from the mean, it is removed from the training set. The removal of examples is made on-line as the architecture is

constructed but a second learning phase with the selected set of examples was implemented as better results can be obtained. In this second phase, the training set is fix as no removal of examples is permitted.

To test the new method for removal of noisy examples a "noise-free" dataset is created from a real dataset, and then controlled noise was added on the attributes (input variables) and on the class (output), separately to see if there was any evident difference between the two cases [19]. The dataset chosen for this analysis is the Pima Indians Diabetes dataset, selected because it has been widely studied and also because it is considered a difficult set with an average generalization ability around 75%. To generate the "noise-free" dataset, the C-Mantec algorithm was run with a single neuron, that classify correctly approximately 70% of the dataset, and then the "noise-free" dataset was constructed by presenting the whole set of inputs through this network to obtain the "noise-free" output. Two different experiments were carried out: in the first one, noise was added to the attributes of the dataset and the performance of the C-Mantec algorithm was analyzed with and without the procedure for noisy examples removal. In Fig. 2 (top) the generalization ability for both mentioned cases is shown for a level of noise between 0 and 0.8 and the results are the average over 100 independent runs. For a certain value of added noise, x, the input values were modified by a random uniform value between $-x$ and x. The bottom graph shows the number of neurons in the generated architectures when the filtering process was and was not applied as a function of the added attribute noise. It can be clearly seen that the removal of the noisy examples help to obtain much more compact architectures and a better generalization ability. The second experiment consisted in adding noise to the output values and the results are shown on Fig. 3. In this case the noise level indicate the probability of modifying the class value to a random value. The results in this case also confirm the effectiveness of the filtering approach in comparison to the case of using the whole "noisy" dataset.

4 Experiments and Results

We tested the noise filtering method introduced in this work using the C-Mantec constructive algorithm on a set of 11 well known benchmark functions. The set of functions contains 6 functions with 2 classes and 5 multi-class problems with a number of classes up to 19. The C-Mantec algorithm was run a maximum number of iterations of 50.000 and an initial temperature (T_0) equals to the number of inputs of the analyzed functions, but it is worth noting that the algorithm is quite robust to changes on these parameters. The results are shown in Table 1, where it is shown the number of neurons of the obtained architectures and the generalization ability obtained, including the standard deviation values, computed over 100 independent runs. The last column of Table 1 shows, as a comparison, the generalization ability values obtained by Prechelt [20] in a work where he analyzed in a systematic way the prediction capabilities of different topologies neural networks, and thus we believe that the reported values are highly optimized. The number and the set of training and test examples were

chosen identically in both compared cases. The results shows that the C-Mantec algorithm outperforms the ones obtained by Prechelt in 6 out of 11 problems and on average the generalization ability is 2.1% larger. Regarding the size of the networks obtained using the method introduced in this work, the architectures are very small for all problems with 2 or 3 classes, for which the architectures contain less than 4 neurons (on average) for all these cases. For the multi-class problems the algorithm generates networks with a larger number of hidden neurons but this is because of the method used to treat multiclass problems that will be reported in [18].

Table 1. Results for the number of neurons and the generalization ability obtained with the C-Mantec algorithm using the data filtering method introduced in this work. The last column shows the results from [20] (See text for more details).

Function	Inputs	Classes	Neurons	Generalization C-Mantec	Generalization NN [20]
Diab1	8	2	3.34 ± 1.11	76.62 ± 2.69	74.17 ± 0.56
Cancer1	9	2	1 ± 0.0	96.86 ± 1.19	97.07 ± 0.18
Heart1	35	2	2.66 ± 0.74	82.63 ± 2.52	79.35 ± 0.31
Heartc1	35	2	1.28 ± 0.57	82.48 ± 3.3	80.27 ± 0.56
Card1	51	2	1.78 ± 0.87	85.16 ± 2.48	86.63 ± 0.67
Mushroom	125	2	1 ± 0.0	99.98 ± 0.04	100.00 ± 0.0
Thyroid	21	3	3 ± 0.0	91.91 ± 0.59	93.44 ± 0.0
H orse1	58	3	3 ± 0.0	66.56 ± 5.08	73.3 ± 1.87
Gene1	120	3	3.03 ± 0.22	88.75 ± 1.07	86.36 ± 0.1
Glass	9	6	17.84 ± 1.19	63.75 ± 6.38	53.96 ± 2.21
Soybean	82	19	171 ± 0.0	91.63 ± 1.89	90.53 ± 0.51
Average	50.27	4.18	18.99 ± 0.43	84.21 ± 2.03	82.50 ± 0.63

5 Discussion

In this work we have introduced a new method for filtering noisy examples using a recently developed constructive neural network algorithm C-Mantec. The filtering method is based on the fact that noisy examples are more difficult to be learnt, and this fact is evident during the learning process in which the constructive algorithm tries to classify correctly the examples using the minimum number of neurons. Noisy examples need more learning updates of the synaptic weights, and this fact permits its identification and further removal. Simulations performed on benchmark datasets show that the generalization ability and size of the resulting network are very much improved after the removal of the noisy examples and a comparison, done against previous reported results [20], shows that the generalization ability was on average a 2.1% larger, indicating the effectiveness of the C-Mantec algorithm implemented with the new filtering stage. It has to be noted that the introduced method of data selection can be used as a pre-processing stage for its use with other prediction algorithms. We have also analyzed the performance of the filtering stage on datasets contaminated by

only attribute or class noise, but did not find any clear difference between these two cases for which the filtering process worked equally well.

Acknowledgements

The authors acknowledge support from CICYT (Spain) through grant TIN2005-02984 (including FEDER funds) and from Junta de Andalucía through grant P06-TIC-01615. Leonardo Franco acknowledges support from the Spanish Ministry of Education and Science through a Ramón y Cajal fellowship.

References

1. Haykin, S.: Neural Networks: A Comprehensive Foundation. Macmillan/IEEE Press (1994)
2. Lawrence, S., Giles, C.L., Tsoi, A.C.: 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 (1996)
3. Gòmez, I., Franco, L., Subirats, J.L., Jerez, J.M.: Neural Networks Architecture Selection: Size Depends on Function Complexity. In: Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E. (eds.) ICANN 2006. LNCS, vol. 4131, pp. 122–129. Springer, Heidelberg (2006)
4. Mezard, M., Nadal, J.P.: Learning in feedforward layered networks: The tiling algorithm. J. Physics A. 22, 2191–2204 (1989)
5. Frean, M.: The upstart algorithm: A method for constructing and training feedforward neural networks. Neural Computation 2, 198–209 (1990)
6. Parekh, R., Yang, J., Honavar, V.: Constructive Neural-Network Learning Algorithms for Pattern Classification. IEEE Transactions on Neural Networks 11, 436–451 (2000)
7. Subirats, J.L., Jerez, J.M., Franco, L.: A New Decomposition Algorithm for Threshold Synthesis and Generalization of Boolean Functions. IEEE Transactions on Circuits and Systems I (in press, 2008)
8. Reed, R.: Pruning algorithms - a survey. IEEE Transactions on Neural Networks 4, 740–747 (1993)
9. Smieja, F.J.: Neural network constructive algorithms: trading generalization for learning efficiency? Circuits, systems, and signal processing 12, 331–374 (1993)
10. do Carmo Nicoletti, M.C., Bertini, J.R.: An empirical evaluation of constructive neural network algorithms in classification tasks. International Journal of Innovative Computing and Applications 1, 2–13 (2007)
11. Bramer, M.A.: Pre-pruning Classification Trees to Reduce Overfitting in Noisy Domains. In: Yin, H., Allinson, N.M., Freeman, R., Keane, J.A., Hubbard, S. (eds.) IDEAL 2002. LNCS, vol. 2412, pp. 7–12. Springer, Heidelberg (2002)
12. Hawkins, D.M.: The problem of Overfitting. Journal of Chemical Information and Computer Sciences 44, 1–12 (2004)
13. Angelova, A., Abu-Mostafa, Y., Perona, P.: Pruning training sets for learning of object categories. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005, vol. 1, pp. 494–501 (2005)

14. Cohn, D., Atlas, L., Ladner, R.: Improving Generalization with Active Learning. Mach. Learn. 15, 201–221 (1994)
15. Cachin, C.: Pedagogical pattern selection strategies. Neural Networks 7, 175–181 (1994)
16. Kinzel, W., Rujan, P.: Improving a network generalization ability by selecting examples. Europhys. Lett. 13, 473–477 (1990)
17. Franco, L., Cannas, S.A.: Generalization and Selection of Examples in Feedforward Neural Networks. Neural Computation 12(10), 2405–2426 (2000)
18. Subirats, J.L., Franco, L., Jerez, J.M.: Competition and Stable Learning for Growing Compact Neural Architectures with Good Generalization Abilities: The C-Mantec Algorithm (in preparation, 2008)
19. Zhu, X., Wu, X.: Class noise vs. attribute noise: a quantitative study of their impacts. Artif. Intell. Rev. 22, 177–210 (2004)
20. Prechelt, L.: Proben 1 – A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms. Technical Report (1994)