Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers and Mathematics with Applications 51 (2006) 527–536
An International Journal
computers & mathematics
with applications
www.elsevier.com/locate/camwa

Determining the Number of Real Roots of Polynomials through Neural Networks

B. MOURRAIN
Project GALAAD, INRIA Sophia Antipolis, 2004 route des Lucioles,
B.P. 93, 06902 Sophia Antipolis, Cedex, France
Bernard.Mourrain@sophia.inria.fr

N. G. PAVLIDIS, D. K. TASOULIS AND M. N. VRAHATIS
Computational Intelligence Laboratory (CI Lab), Department of Mathematics
University of Patras, GR-26110 Patras, Greece
and
University of Patras Artificial Intelligence Research Center (UPAIRC),
University of Patras, GR-26110 Patras, Greece
<npav><dtas><vrahatis>@math.upatras.gr

(Received and accepted July 2005)

Abstract—The ability of feedforward neural networks to identify the number of real roots of univariate polynomials is investigated. Furthermore, their ability to determine whether a system of multivariate polynomial equations has real solutions is examined on a problem of determining the structure of a molecule. The obtained experimental results indicate that neural networks are capable of performing this task with high accuracy even when the training set is very small compared to the test set.

Keywords—Roots of polynomials, Neural networks, Number of zeros.

1. INTRODUCTION

Numerous problems in mathematical physics, robotics, computer vision, computational geometry, signal processing etc., involve the solution of polynomial systems of equations. Recently, artificial feedforward neural networks (FNNs) have been applied to the problem of computing the roots of a polynomial [1–3]. The underlying idea to construct an FNN capable of finding the roots is to factorize the polynomial into many subfactors on the outputs of the hidden layer of the network. The connection weights from the input layer to the hidden layer are then trained using a suitable algorithm. Thus, the connection weights of the trained network are the roots of the underlying polynomial [1]. A crucial advantage of this approach is that all the roots are obtained simultaneously and in parallel. As traditional root-finding methods identify roots sequentially,

We wish to thank the editor and the anonymous reviewers for their useful comments and suggestions. This work was pursued during the collaboration program CALAMATA (Equipe associée INRIA-GRECE).

0898-1221/06/$ - see front matter Typeset by AMS-TEX
doi:10.1016/j.camwa.2005.07.012

that is, the next root is obtained by the deflated polynomial after the former root is found, their accuracy is fundamentally limited and cannot surpass that of an FNN [4]. Furthermore, increasing the number of processors will not increase the speed of these algorithms, as they are inherently sequential.

In [3], FNNs were applied first to the factorization of 2D second-order polynomials. To this end, a constrained learning algorithm which incorporates *a priori* information about the problem into the backpropagation algorithm was proposed [3,5]. The results indicate that the constrained learning algorithm not only exhibits rapid convergence compared to the standard backpropagation algorithm, but also yields more accurate results. Inspired by this approach a method for finding the roots of a polynomial using a constrained learning algorithm that imposes as constraints the relationships between the roots and the coefficients of the polynomial, was proposed in [6]. The most computationally demanding step of this method is the computation of the constraint conditions and their derivatives. The estimated cost for these computations is of the order $\mathcal{O}(2^n)$, where n is the order of the polynomial. This exponential complexity renders the particular method impractical for high-order polynomials [1]. To overcome this limitation, another constrained learning algorithm based on the relationships between root moments and the polynomial coefficients was proposed. The estimated computational cost of computing the set of constraints and their derivatives for the latter method is $\mathcal{O}(mn^3)$, where n is as before the order of the polynomial, and m is the number of constraints used by the learning algorithm [4]. Exploiting a recursive root moment method, the computational complexity can be reduced to $\mathcal{O}(mn^2)$ [4]. Note that the computational complexity of traditional root-finding methods such as Muller and Laguerre is of the order $\mathcal{O}(3^n)$, while that of the fastest methods, like Jenkins-Traub, is of the order $\mathcal{O}(n^4)$. In [4], these approaches were extended to the more general problem of finding arbitrary (including real or complex) roots of arbitrary polynomials. The experimental results reported in [1] indicate that on the task of computing all the roots of a polynomial, the FNNs trained through constrained learning algorithms are both faster and more accurate than traditional Muller and Laguerre methods. Furthermore, the constrained learning algorithm is not sensitive to the initialization of the weights (roots) of the network.

To the best of our knowledge the problem of computing the number of real roots of a univariate polynomial, as well as, that of determining the existence of real solutions of a system of multivariate polynomial equations, have not been addressed through neural computation approaches. FNNs are considered to be powerful classifiers compared to classical algorithms such as the nearest neighbor method. The algorithms used in FNNs are capable of finding a good classifier based on a limited, and in general small, number of training examples. This capability, also referred to as generalization, is of particular interest in classification tasks. In this paper, we train FNNs to determine the number of real roots of univariate polynomials using as inputs the values of the coefficients. Next, we investigate their ability to accurately identify the number of real roots for combinations of coefficients not encountered during training. Our findings suggest that FNNs are highly accurate on this task even when the training set is very small in proportion to the test set. Subsequently, we employ a system of multivariate polynomial equations, and investigate the ability of FNNs to discriminate between combinations of coefficients that yield only complex roots and those that also yield real solutions. This appears to be a much harder problem for FNNs, but the trained networks are able to attain a satisfactory performance.

The rest of the paper is organized as follows. Section 2 briefly introduces FNNs and discusses some of their theoretical properties. Section 3 is devoted to the presentation of the experimental results obtained. Finally, conclusions and directions for future research are provided in Section 4.

2. ARTIFICIAL NEURAL NETWORKS

FNNs are parallel computational models comprised of densely interconnected, simple, adaptive processing units, characterized by an inherent propensity for storing experiential knowledge and

rendering it available for use. FNNs resemble the human brain in two fundamental respects. First, knowledge is acquired by the network from its environment through a learning process. Second, interneuron connection strengths, known as synaptic weights are employed to store the acquired knowledge [7]. The structure of FNNs enables them to learn highly nonlinear relationships and adapt to changing environments. Among the highly desirable features of FNNs is their capability to handle *incompleteness*, i.e., missing parameter values; *incorrectness*, i.e., systematic, or random noise in the data; *sparseness*, i.e., few and/or nonrepresentable records; and *inexactness*, i.e., inappropriate selection of parameters for the given task. These characteristics render FNNs capable of finding a good classifier based on a limited number of training examples.

In FNNs neurons are organized in layers and no feedback connections are present. Inputs are assigned to the sensory neurons, which form the input layer of the network, while outputs are obtained by the neurons of the final layer, also called the output layer. All other neurons are organized in the intermediate layers, which are called hidden layers. This structure allows the representation of an FNN with a series of integers. For example, with x-y-z we refer to an FNN with x input neurons, a single hidden layer consisting of y neurons, and an output layer containing z neurons. Inputs to the network are assigned to the input neurons and after the computations at each layer are completed the outputs are propagated to the subsequent layer. The output of the network is the outcome of the computations of the output layer neurons.

The operation of an FNN is based on the following equations that describe the workings of the j^{th} neuron at the l^{th} layer of the network,

$$\text{net}_j^l = \sum_{i=1}^{n_{l-1}} w_{ij}^{l-1,l} y_i^{l-1} + \theta_j^l, \qquad y_j^l = f\left(\text{net}_j^l\right), \tag{1}$$

where net_j^l is the sum of the weighted inputs of the j^{th} neuron in the l^{th} layer, where $j = 2, \ldots, O$. The additional term θ_j^l denotes the bias of this neuron. The weighted sum net_j^l is called the excitation level of the neuron [7]. The weight connecting the output of the i^{th} node at the $(l-1)$ layer to the j^{th} neuron at the l^{th} layer is denoted by $w_{ij}^{l-1,l}$. Finally, y_j^l is the output of the j^{th} neuron of the l^{th} layer, and $f(\text{net}_j^l)$ is the activation function of that neuron.

In supervised training there is a fixed, finite set of input-output samples (patterns) that are employed by the training procedure to adjust the weights of the network. Assuming that there are P input-output samples, the squared error over the training set is defined as

$$E(w) = \sum_{p=1}^{P} \sum_{j=1}^{n_O} \left(y_{jp}^O - t_{jp}\right)^2 = \sum_{p=1}^{P} \sum_{j=1}^{n_O} \left[f^O\left(\text{net}_j^O\right) - t_{jp}\right]^2, \tag{2}$$

where, n_O stands for the number of neurons at the output layer of the network, y_{jp}^O stands for the output of the j^{th} output neuron when the input to the network was the p^{th} training pattern, and t_{jp} denotes the j^{th} desired response for the p^{th} training pattern. Equation (2) is called the *error function* of the network, and the purpose of training is to yield a set of network weights that will minimize it. It should be noted at this point that any distance function, such as the Minkowsky, Mahalanobis, Camberra, Chebychev, quadratic, correlation, Kendall's rank correlation and Chi-square distance metrics; the context-similarity measure; the contrast model; hyperectangle distance functions and others [8], can be used in the error function.

The efficient supervised training of FNNs, which amounts to the minimization of the error function, is a subject of considerable ongoing research and a number of efficient and effective algorithms have been proposed in literature [9–18].

Two crucial parameters for the successful application of FNNs on any problem, are the selection of appropriate network architecture and training algorithm. The problem of identifying the optimal network architecture for a specific task remains up to date an open and challenging problem. For the general problem of function approximation, the *universal approximation theorem* [19–21] states the following.

THEOREM 1. *Standard feedforward networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function to any desired degree of accuracy.*

An immediate implication of the above theorem is that any lack of success in applications must arise either due to inadequate learning, or an insufficient number of hidden units, or the lack of a deterministic relationship between the inputs and the targets. A second theorem proved in [22] provides an upper bound for the architecture of an FNN destined to approximate a continuous function defined on the hypercube in $\mathbb{R}^n$.

THEOREM 2. *On the unit cube in $\mathbb{R}^n$ any continuous function can be uniformly approximated, to within any error by using a two hidden layer network having $2n+1$ units in the first layer and $4n+3$ units in the second layer.*

3. RESULTS

In the present study, we investigate the ability of FNNs to determine the number of real roots of polynomials. In more detail, for polynomials of a specific degree, we construct a series of combinations of the values of the coefficients. We solve these polynomials and determine the number of real roots for each coefficient combination. Thus, we construct datasets with the values of the coefficients and the number of real roots corresponding to these coefficient values. We split these datasets into two parts, a training set and a test set, and employ the patterns belonging to the training set to perform the supervised training of the FNNs. As input to the FNN we supply the values of the coefficients, while the desired output (target) is the number of real roots. After the training procedure has adjusted the weights of the network, we investigate its ability to correctly identify the number of real roots for combinations of the values of the coefficients that the FNN has not previously encountered. In other words, we evaluate its classification ability on the test set.

To compute the number of real roots of the polynomials, we employed routines included in the symbolic numeric applications (SYNAPS 2.1.2) library [23]. One of the methods that we used for solving the univariate equations is a subdivision solver, based on Descartes rule. The univariate polynomial is expressed into the Bernstein basis and the domain is subdivided until the number of sign changes of the coefficients is 0 or 1, or until a given precision ε is reached. This yields isolating intervals containing one root if the root is simple and its multiplicity (up to a perturbation ε) otherwise. The interest of the method is its speed and the certification for well-separated simple roots. For more details on this method, see [24].

The other method that we considered is called Aberth's method. It is an extension of Weierstrass method, which consists of applying Newton's iteration to the square multivariate system connecting the roots with the coefficients of a univariate polynomial. This iteration converges to a vector that contains all the roots of the polynomial. We use the implementation by Bini and Fiorantino provided in SYNAPS [23]. This method yields the complex roots, from which we extract the real roots by using a threshold ε on the imaginary part. The interest of this method is the control of the error, even in the case of multiple roots. See [25] for more details.

Next, we proceed with the description of the datasets employed in the present study. For univariate polynomials of degree two to four, all coefficients were allowed to assume integer values in the range $[1, 10]$. For the fifth-, sixth-, and seventh-degree polynomials, integer coefficients in $[-3, 3]$, $[-6, 6]$, and $[-6, 6]$, respectively, were considered. The data sets used for training and testing were constructed by taking all the permutations of the coefficients in the aforementioned ranges. The only exceptions to this rule were for the sixth- and seventh-degree polynomials for which a total of 218748 and 475938, respectively, random permutations of the coefficients were constructed.

For the polynomials of degree two to four, approximately two-thirds of the total patterns were used for training the networks, while the remaining one-third was used to evaluate the

generalization performance. The obtained results suggested that the generalization capability of the trained networks was not significantly inhibited by a reduction of the size of the training set. To this end, we employed substantially smaller training sets for the fifth- and sixth-degree polynomials. In particular, for the fifth-degree polynomials, only 900 patterns were used for training and the remaining 99144 patterns were assigned to the test set. For the sixth-degree polynomials, the training set consisted of 1250 patterns, while the test set contained 217498 patterns. Similarly, for the seventh-degree polynomials, the training set was comprised of 838 patterns, while the test set contained 475100 patterns.

The problem of selecting the optimal network architecture for a particular task remains up-to-date an open problem. In this work we employed FNNs with two hidden layers and architecture Z-8-7-Y, where Z stands for the number of coefficients of the polynomial and Y represents the number of classes in each case. To train the networks, we employed three well established batch training algorithms, and an on-line training algorithm; namely, the resilient propagation algorithm (RPROP) [14], the improved resilient propagation algorithm (iRPROP) [10], the scaled conjugate gradient method (SCG) [18], and the adaptive on-line backpropagation algorithm (AOBP) [11]. All methods were allowed to perform 500 epochs, and for each method 100 experiments were performed. The parameters employed by the methods were set to the values suggested in the references [10,11,14,18]. The scope of this work is to investigate the capability of FNNs to address the problem of determining the number of real roots, and not to provide an extensive review of the performance of different training methods. We intend to pursue this issue in a future correspondence.

For univariate polynomials of degree two to five the choice of training algorithm did not bear a significant impact on the resulting performance. The obtained results from an indicative experiment for these cases are reported in truth Tables 1 and 2. Each table reports the number of

Table 1.

Polynomials of degree 2. Class 1: Zero real roots. Class 2: Two real roots.			
	Training Set Performance		
	Class 1	Class 2	C.A. (%)
Class 1	542	4	99.267
Class 2	0	154	100
	Test Set Performance		
Class 1	246	1	99.595
Class 2	1	52	98.1132

Polynomials of degree 3. Class 1: One real root. Class 2: Three real roots.			
	Training Set Performance		
	Class 1	Class 2	C.A. (%)
Class 1	5951	2	99.966
Class 2	7	40	85.106
	Test Set Performance		
Class 1	3930	2	99.949
Class 2	8	36	81.818

Table 2.

Polynomials of degree 4. Class 1: Zero real roots. Class 2: Two real roots.			
	Training Set Performance		
	Class 1	Class 2	C.A. (%)
Class 1	18621	113	99.396
Class 2	227	41039	99.449
	Test Set Performance		
Class 1	11870	109	99.090
Class 2	151	27121	99.446

Polynomials of degree 5. Class 1: One real root. Class 2: Three real roots. Class 3: Five real roots.				
	Training Set Performance			
	Class 1	Class 2	Class 3	C.A. (%)
Class 1	396	4	0	99
Class 2	3	397	0	99.25
Class 3	0	3	97	97
	Test Set Performance			
Class 1	61908	5563	133	91.57
Class 2	2916	27087	1023	87.30
Class 3	8	89	425	82.68

correctly classified and misclassified patterns, as well as the resulting classification accuracy (C.A.), for both the training set and the test set. As shown in Tables 1 and 2, FNNs were successfully trained to identify the number of real roots of polynomials of degree two to five, achieving classification accuracies around 99% on the training set. The only exception to this behavior was witnessed for third-degree polynomials with three real roots (as shown in the right part of Table 1), but this can be attributed to the very small representation of such polynomials in the training set (47 out of 6000 patterns). The most important finding, however, is that the trained FNNs exhibited a generalization capability very close to their performance on the training set. Even for the class of third degree polynomials with three real roots, the trained FNNs exhibited a generalization performance of 81.818%.

On the other hand, for univariate polynomials of degree six and seven the choice of the training algorithm bore a substantial impact on performance. To allow a better visualization of the performance on the test set, we present the results using boxplots. A boxplot is a diagram that conveys location and variation information about a certain variable. The median classification accuracy is displayed as a horizontal line and a box is drawn between the first and third quartile of observations. Then, the minimum and maximum values that lie into the range with center the median and length 1.5 multiplied by the interquartile range are connected to the box. If a value lies outside this range, then it is considered as an outlier and displayed as a dot. Notches represent a robust estimate of the uncertainty about the median.

As illustrated in Figure 1, for polynomials of degree six the best performing method was iRPROP. This method attained the highest median performance, for polynomials with zero, two, and four real roots (classes 1, 2, and 3). It also exhibited the most robust performance for these

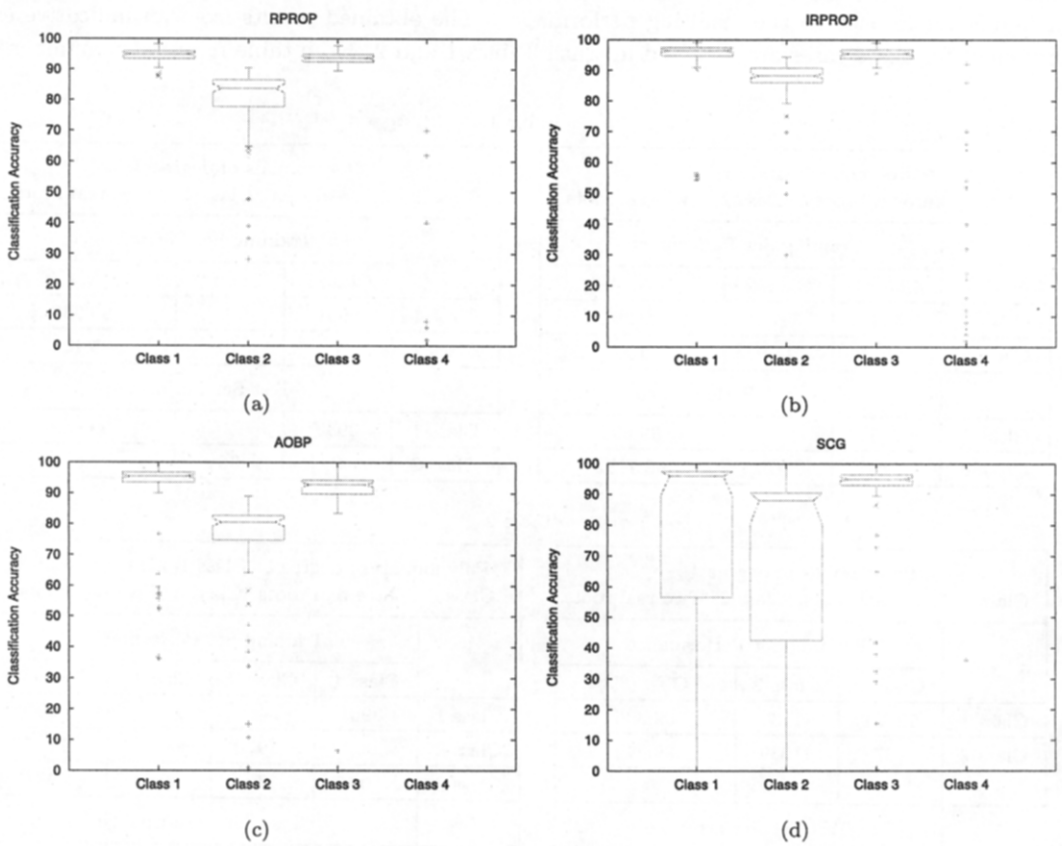

Figure 1. Polynomials of degree 6. Class 1: Zero real roots; Class 2: Two real roots; Class 3: Four real roots; Class 4: Six real roots.

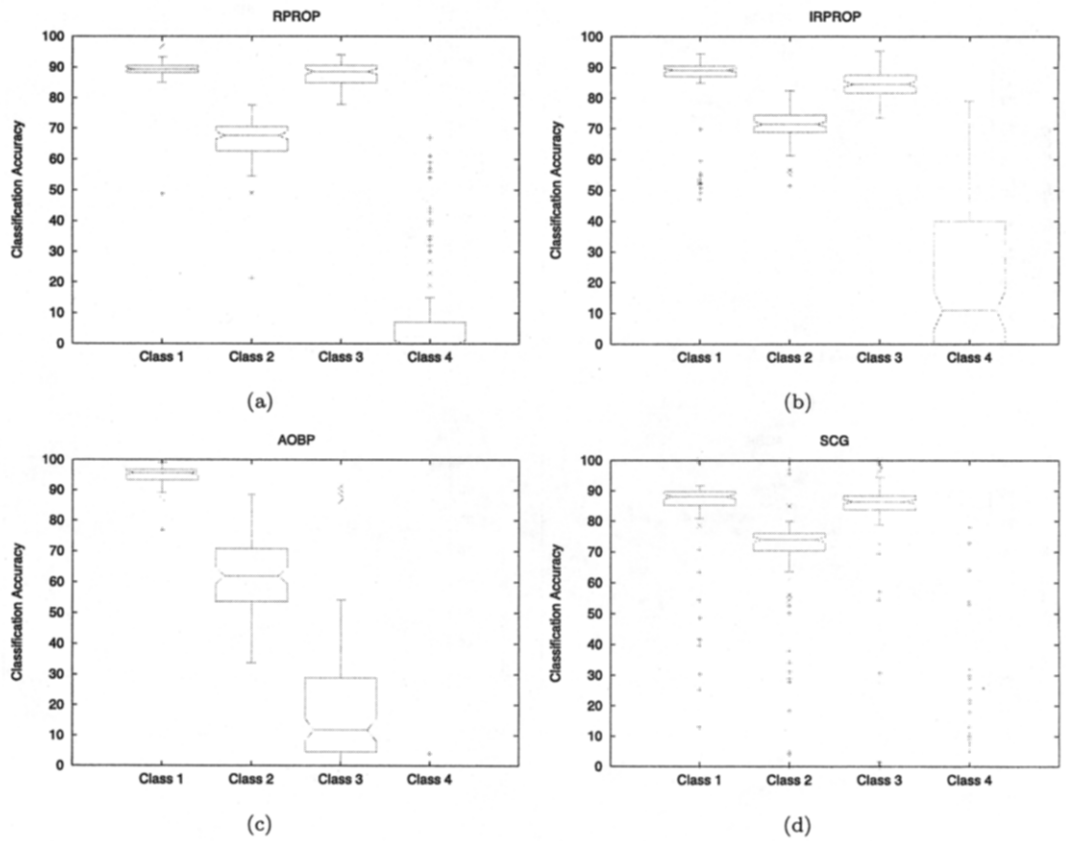

Figure 2. Polynomials of degree 7. Class 1: One real root; Class 2: Three real roots; Class 3: Five real roots; Class 4: Seven real roots.

classes, as suggested by the width of the boxes. For polynomials with six real roots, no method proved capable of achieving a good classification performance on average. This finding can be attributed to the relatively small representation of this class of polynomials in the training set (50 patterns out of 1250). iRPROP and RPROP were the only methods that managed to train networks that achieved a high classification accuracy for this class. Overall, the performance of the RPROP method was close to that of iRPROP, while AOBP performed slightly worse than the two previously mentioned methods. The worst performing method was SCG, whose performance varied greatly for polynomials of class one and two, despite the fact that its median performance was high for these classes.

The results for the seventh-degree univariate polynomials, illustrated in Figure 2, suggest that iRPROP was the best performing and most robust method. As in the case of sixth-degree polynomials, the trained networks misclassified polynomials belonging to class four, that is polynomials with seven real roots. Once again, the representation of this class in the data set was very small. iRPROP was the only considered method that exhibited a median performance higher than zero for this class. The performance of SCG and RPROP were close to that of iRPROP. In this case, the worst performing method was AOBP that exhibited a very low classification accuracy on the third class, and a relatively volatile performance on class two.

The reported results for polynomials of degree five, six, and seven support the claim that even when the number of training patterns is greatly smaller than that of the test patterns the trained FNNs manage to attain a high classification accuracy on the test set.

At a next step we tested this approach on a system of multivariate polynomials with a given support. In detail, we used the system of polynomials exhibited in equation (3). This set of polynomials describes the six atom molecule problem. More specifically, the six-atom molecule

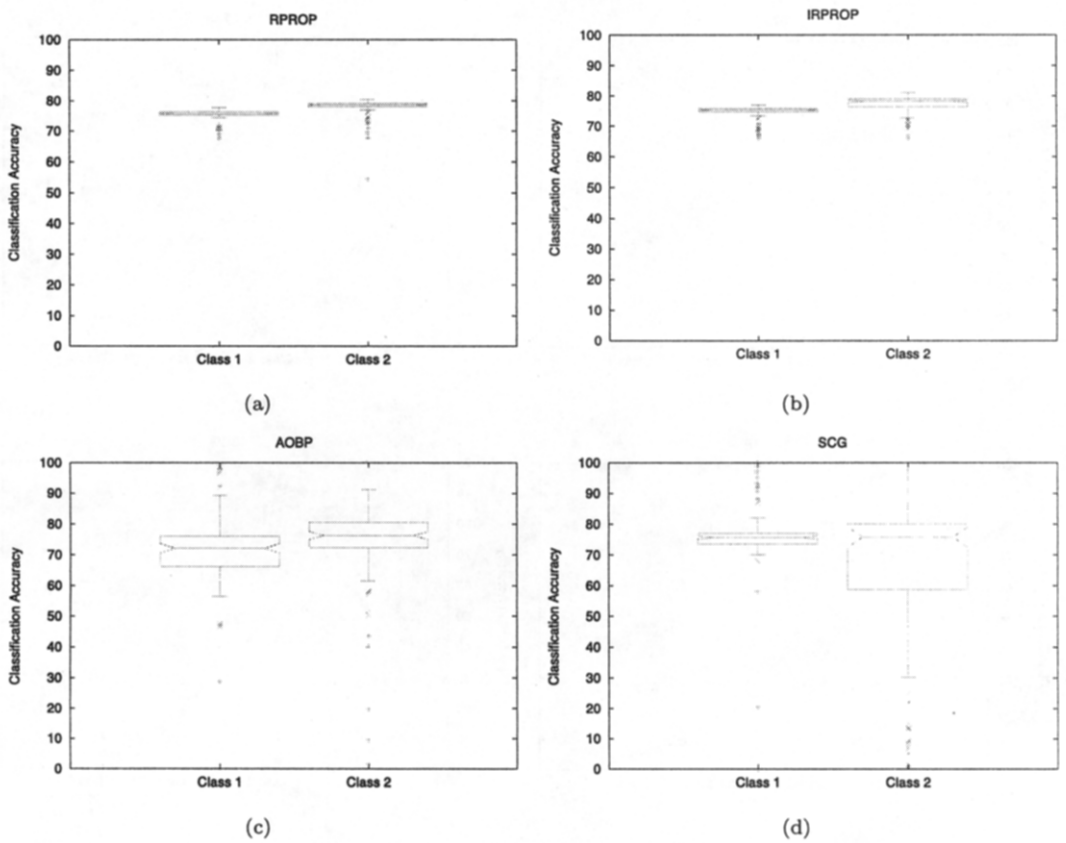

Figure 3. Results for the six atom molecule problem. Class 1: No real solutions; Class 2: Real solutions exist.

problem amounts to finding the global geometry of the molecule, knowing the lengths between the atoms and the angles between two consecutive links. It is known that the problem can be modeled with the following three polynomial equations of three variables,

$$\begin{aligned} f_1 &= \beta_{11} + \beta_{12}t_2^2 + \beta_{13}t_3^2 + \beta_{14}t_2t_3 + \beta_{15}t_2^2t_3^2 = 0, \\ f_2 &= \beta_{21} + \beta_{22}t_3^2 + \beta_{23}t_1^2 + \beta_{24}t_3t_1 + \beta_{25}t_3^2t_1^2 = 0, \\ f_3 &= \beta_{31} + \beta_{32}t_1^2 + \beta_{33}t_2^2 + \beta_{34}t_1t_2 + \beta_{35}t_1^2t_2^2 = 0. \end{aligned} \tag{3}$$

It is known further that there are at most 16 isolated solutions to this problem. For this problem, we constructed 45000 real valued random combinations for the coefficients β_{ij}, from a uniform distribution in the range $[-10, 10]$. The FNNs were employed to determine whether a combination of coefficients results in a polynomial system that has solely complex roots, or whether real roots also exist. Thus, a coefficient combination is assigned to Class 1 if all the solutions of the corresponding polynomial system are complex, and to Class 2 if real solutions exist. From the 45000 combinations 9000 were used for training and 35000 for testing. The topology of the FNN employed was 15-8-7-2. Figure 3 illustrates the results obtained for the four different methods over 100 runs for each algorithm.

On this test problem the best performing methods were RPROP and iRPROP, with RPROP being slightly more robust. Both these methods attained a median classification accuracy close to 80% for both classes. AOBP achieved a lower median performance for Class 1. On the other hand, it attained the highest maximum classification accuracy among all the considered methods. Finally, SCG attained slightly worse performance to RPROP and iRPROP for Class 1, but its performance is much worse on Class 2, in which there is a significant variation of performance.

4. DISCUSSION AND CONCLUDING REMARKS

In the present paper, we investigated the ability of FNNs to determine the number of real roots of univariate polynomials. To this end, we considered three well known and widely used batch training algorithms, and an on-line training algorithm. The experimental results suggest that FNNs are capable of accurately classifying the number of real roots of low-degree univariate polynomials using as input the coefficients. Most importantly, the considered FNNs exhibited a very high generalization ability, even when the size of the training set was very small compared to that of the test set. For polynomials of degree two to five the choice of training algorithm did not bear a significant impact on the resulting generalization ability. Differences were witnessed, however, for the sixth-degree and seventh-degree polynomials. For these polynomials, among the four considered training algorithms the resilient propagation and the improved resilient propagation algorithms exhibited the highest and most robust classification accuracies. The classes that corresponded to the six and seven real roots, for the six- and seven-degree polynomials respectively, were marginally represented in the dataset. For these classes all the training algorithms exhibited a very low, and in most cases zero, median generalization ability. Among the methods considered, the only ones that were capable of training networks that yielded high classification accuracies with respect to these classes were the resilient propagation and the improved resilient propagation.

Training feedforward neural networks to determine if the system of multivariate polynomial equations corresponding to the six atom molecule problem has real solutions for a random combination of coefficients, proved to be a more difficult task. The trained networks achieved a lower training and generalization ability in comparison to the cases of univariate polynomials. However, even in this case a generalization accuracy close to 80% was achieved using only a small portion of the dataset as training set. In a future correspondence, we intend to perform a thorough investigation of the performance of FNNs on higher degree univariate polynomials, as well as, systems of multivariate polynomial equations, using an extensive range of training algorithms.

REFERENCES

1. D.S. Huang, H.H.S. Ip and Z. Chi, A neural root finder of polynomials based on root moments, *Neural Computation* **16** (8), 1721–1762, (2004).
2. D.S. Huang and C. Zheru, Neural networks with problem decomposition for finding real roots of polynomials, In *Proceedings of the International Joint Conference on Neural Networks 2001 (IJCNN'01)*, July 15–19, pp. 25–30, Washington, DC, (2001).
3. S.J. Perantonis, N. Ampazis, S. Varoufakis and G. Antoniou, Constrained learning in neural networks: Application to stable factorization of 2-D polynomials, *Neural Processing Letters* **7**, 5–14, (1998).
4. D.S. Huang, A constructive approach for finding arbitrary roots of polynomials by neural networks, *IEEE Transactions on Neural Networks* **15** (2), 477–491, (2004).
5. D.A. Karras and S.J. Perantonis, An efficient constrained learning algorithm for feedforward networks, *IEEE Transactions on Neural Networks* **6**, 1420–1434, (1995).
6. D.S. Huang, Constrained learning algorithms for finding the roots of polynomials: A case study, In *Proc. IEEE Region 10 Tech. Conf. on Computers, Communications, Control and Power Engineering*, pp. 1516–1520, (2002).
7. S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, New York, (1999).
8. D.R. Wilson and T.R. Martinez, Improved heterogeneous distance functions, *Journal of Artificial Intelligence Research* **6**, 1–34, (1997).
9. M.T. Hagan and M. Menhaj, Training feedforward networks with the marquardt algorithm, *IEEE Transactions on Neural Network* **5** (6), 989–993, (1994).
10. C. Igel and M. Hüsken, Improving the Rprop learning algorithm, In *Proceedings of the Second International ICSC Symposium on Neural Computation (NC 2000)*, (Edited by H. Bothe and R. Rojas), pp. 115–121, ICSC Academic Press, (2000).
11. G.D. Magoulas, V.P. Plagianakos and M.N. Vrahatis, Adaptive stepsize algorithms for on-line training of neural networks, *Nonlinear Analysis, Theory, Methods and Applications* **47**, 3425–3430, (2001).
12. G.D. Magoulas, M.N. Vrahatis and G.S. Androulakis, Effective backpropagation training with variable stepsize, *Neural Networks* **10** (1), 69–82, (1997).
13. G.D. Magoulas, M.N. Vrahatis and G.S. Androulakis, Increasing the convergence rate of the error backpropagation algorithm by learning rate adaptation methods, *Neural Computation* **11** (7), 1769–1796, (1999).

14. M. Riedmiller and H. Braun, A direct adaptive method for faster backpropagation learning: The rprop algorithm, In *Proceedings of the IEEE International Conference on Neural Networks*, pp. 586–591, San Francisco, CA, (1993).
15. M.N. Vrahatis, G.S. Androulakis, J.N. Lambrinos and G.D. Magoulas, A class of gradient unconstrained minimization algorithms with adaptive stepsize, *J. Comput. Appl. Math.* **114** (2), 367–386, (2000).
16. M.N. Vrahatis, G.D. Magoulas and V.P. Plagianakos, Globally convergent modification of the quickprop method, *Neural Processing Letters* **12**, 159–170, (2000).
17. M.N. Vrahatis, G.D. Magoulas and V.P. Plagianakos, From linear to nonlinear iterative methods, *Applied Numerical Mathematics* **45** (1), 59–77, (2003).
18. M. Møller, A scaled conjugate gradient algorithm for fast supervised learning, *Neural Networks* **6**, 525–533. (1993).
19. G. Cybenko, Approximations by superpositions of sigmoidal functions, *Mathematics of Control, Signals, and Systems* **2**, 303–314, (1989).
20. K. Hornik, Multilayer feedforward networks are universal approximators, *Neural Networks* **2**, 359–366, (1989).
21. H. White, Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings, *Neural Networks* **3**, 535–549, (1990).
22. A. Pinkus, Approximation theory of the mlp model in neural networks, *Acta Numerica*, 143–195, (1999).
23. G. Dos Reis, B. Mourrain, F. Rouillier and Ph. Tribuchet, An environment for symbolic and numeric computation, Technical Report ECG-TR-122102-03, INRIA, Sophia-Antipolis, (2002).
24. B. Mourrain, M.N. Vrahatis and J.C. Yakoubsohn, On the complexity of isolating real roots and computing with certainty the topological degree, *Journal of Complexity* **18** (2), 612–640, (2002).
25. D. Bini, Numerical computation of polynomial zeros by means of Aberth's method, *Numerical Algorithms* **13**, 179–200, (1996).