A Constructive Approach for Finding Arbitrary Roots of Polynomials by Neural Networks

De-Shuang Huang, *Senior Member, IEEE*

***Abstract*—This paper proposes a constructive approach for finding arbitrary (real or complex) roots of arbitrary (real or complex) polynomials by multilayer perceptron network (MLPN) using constrained learning algorithm (CLA), which encodes the *a priori* information of constraint relations between root moments and coefficients of a polynomial into the usual BP algorithm (BPA). Moreover, the root moment method (RMM) is also simplified into a recursive version so that the computational complexity can be further decreased, which leads the roots of those higher order polynomials to be readily found. In addition, an adaptive learning parameter with the CLA is also proposed in this paper; an initial weight selection method is also given. Finally, several experimental results show that our proposed neural connectionism approaches, with respect to the nonneural ones, are more efficient and feasible in finding the arbitrary roots of arbitrary polynomials.**

***Index Terms*—Adaptive learning parameters, computational complexity, constrained learning, multilayer perceptron networks, polynomials, recursive, root moment, roots**

I. Introduction

THE roots (zeros) finding of polynomials is often required by various areas of signal processing, such as speech processing, frequency or direction-of-arrival (DOA) estimation of signals in noise, filter design, spectral analysis, phase unwrapping, communication (coding theory and decoding theory, cryptography), etc., [1]–[8]. So far there have existed many reports on finding the roots of polynomials for a variety of signal processing topics. Most of them, however, adopted a conventional numerical method of roots finding, such as Laguerre's, Newton–Raphson's, and Jenkins–Traub's methods, etc., [9], [10], in which successive approximations to the roots are obtained [2], [9]–[13]. Moreover, almost all numerical methods can only find the roots one by one, i.e., by *deflation* method, the next root is obtained by the deflated polynomial after the former root is found [9]. This means that the numerical root-finding algorithm is inherently sequential.

Neural networks, especially multilayer perceptron network (MLPN), have been used successfully in many fields [14]–[24]. Some examples include using two-layered perceptron network to do the factorization of polynomials with two or multiple variables [16], [17], applying one-layered linear perceptron for the inversion of nonsingular matrices [21] and solving linear equations by neural networks [22], [23], etc. These successful applications show that neural networks can indeed solve many (neural) computation problems of signal processing related to linear algebra. The conventional BP algorithm (BPA) with a gradient descent type, however, which often exhibits slow convergence, largely limits their wide use in more neural computation fields. For example, in 2000, Huang designed a Sigma-Pi (Σ-Π) structured neural network to find the real roots of real coefficient polynomials [24]. It was found that the BP training algorithm would spend much time to converge unless those initial values of network synapse weights, corresponding to the root values, are appropriately selected.

In 1995, Perantonis, *et al.* proposed, using constrained learning (CL) technique, which incorporated the *a priori* information from problems into the conventional BPA to train MLPNs [14], [15] and to perform factorization of 2-D polynomials [16]. The results show that the constrained learning algorithm (CLA) not only exhibits rapid convergence, but also renders accurate computation values. Hence, it is an effective and practical method for solving those problems with constrained conditions. Inspired by this training approach, we can solve the above mentioned finding roots of polynomials by using the CLA, which imposes the constrained relation between the roots and coefficients of a polynomial into the CLA to train a suitably structured MLPN. Some encouraging computer simulation results were reported [25]. This method, however, will still take much training time for high–order polynomials because of computing the constraint relation involved in between the roots and the polynomial coefficients. Therefore, to shorten the long training time in finding the roots of high–order polynomials, another constrained learning method based on the constrained relation between the root moments [26] and the polynomial coefficients is proposed. Specifically, this root–moment approach can be simplified into a compact recursive version so that the computational complexity can be further lowered. As a result, those problems of rooting high–order polynomials can be easily solved.

In this paper, we further extend these results into a more general case of finding arbitrary (including real or complex) roots of arbitrary (including real or complex) polynomials. In the process of following derivations, we will always consider the case of finding complex roots of polynomials. Specifically, we also give an adaptive method of how to select the learning parameters with the CLAs and an initial weight selection method for the root finders.

Why do we revisit this topic of the root-finding polynomial since there have been many numerical methods? The motivation based on three facts is stated as follows.

1) It is well known that neural networks with the flexible parallel structures like brain can obtain *simultaneously* the problem solutions. Therefore, by using neural network approach to find the roots of polynomials, we can obtain

Manuscript received April 11, 2002; revised April 24, 2003 and December 5, 2003. This work was supported by a grant from NSF of China and the Grant of "100 Persons Program" of Chinese Academy of Sciences of China.

The author is with the Institute of Intelligent Machines, Chinese Academy of Sciences, P.O. Box 1130, Hefei, Anhui 230031, China (e-mail: dshuang@iim.ac.cn; huangdeshuang@yahoo.com).

Digital Object Identifier 10.1109/TNN.2004.824424

all roots *simultaneously and in parallel*, in particular, when the computations were run on a parallel computing machine. On the contrary, most of nonneural numerical methods can only find the roots one by one and increasing the number of processors will not increase the speed of finding the solution. Therefore, in essence, the speed for the numerical methods is certainly slower than the neural methods.

2) Since the numerical methods are by *deflation* method, to sequentially find the roots, each new root is known with only finite accuracy and errors will creep into the determination of the coefficients of the successively deflated polynomial [9]. Hence, the accuracy for the nonneural numerical approach is fundamentally limited and is not possible to surpass the one for the neural root-finding approach that *simultaneously* finds all roots.
3) All nonneural approaches need to find the good candidates of initial root values, otherwise the designed root finder will not converge [9], [10], which will certainly increase the computational complexity and consume additional processing time. On the other hand, however, our proposed neural approaches are instructed by the *a priori* information from the problem imposed into the CLA so that they *almost* do not need to compute any initial root values except for randomly selecting them from the uniform distribution in $[-1, 1]$. The initial weight selection method (IWSM) discussed in the sequel, however, is only done if we wish to get some insights into the initial weights so that the selected weights are capable of assuring the more rapid training to be achieved. Specifically, an important point to be stressed is that the IWSM is adopted only if the consumed time for selecting the initial weights can be neglected with respect to the one for training the neural root–finder involved.

So it is in considering the three facts above that we address connectionism method to discuss this topic of root-finding polynomial.

This paper is organized as follows. Section II presents the fundamental problem of finding the roots of polynomials, including the constrained relations implicit in polynomials such as the roots-coefficients relation and the root-moments coefficients relation. Section III discusses the MLPN root finder model for rooting polynomials and presents the corresponding complex CLAs. Section IV focuses on the computational complexity estimates of the root-moment method, the recursive root-moment approach, and the IWSM as well as the adaptive learning parameter scheme. Section V presents and discusses several experimental results. Finally, several concluding remarks and further research directions are arranged in Section VI.

II. Problem Presentation

A. n-Order Arbitrary Polynomial Problem

For a given n-order polynomial $f(z) \in C$ (C denotes the complex domain)

$$f(z) = a_0 z^n + a_1 z^{n-1} + a_2 z^{n-2} + \cdots + a_{n-1} z + a_n = \sum_{k=0}^{n} a_{n-k} z^k. \tag{1}$$

Without losing generality, letting $a_0 = 1$ and $n \geq 2$, we can write the corresponding complex polynomial coefficients $a_i (i = 0, 1, \ldots, n-1)$ into a vector form $\mathrm{a} = [a_0, a_1, \ldots, a_{n-1}]^T \in \mathrm{C^n}$. The aim of finding the roots of this polynomial is how to obtain the root vector $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_n]^T \in \mathrm{C^n}$ so that the polynomial can be factorized into $f(z) = \prod_{i=1}^{n} (z - \lambda_i)$. In the past, one generally used numerical methods to solve this problem [2], [9]–[11]. Here, we consider adopting neural network technique to seek the solutions. Assume that we construct an MLPN to deal with this problem of finding complex roots of polynomial; then, the n-approximate complex roots $w_i (i = 1, 2, \ldots, n)$ are estimated from the network involved. As a result, we can obtain

$$f(z) \approx \prod_{i=1}^{n} (z - w_i). \tag{2}$$

For this n-order polynomial equation $f(z) = 0$ with n complex roots $w_i (i = 1, 2, \ldots, n)$, the problem is how an MLPN model can be designed to find these complex roots. In the following section, we will present the basic principle using the MLPN model to find complex roots of polynomial.

B. Constrained Relations Implicit in Polynomials

Before presenting a new complex CLA by imposing the constrained relations from the problem into the conventional BPA, it is necessary for us to investigate the constrained relations or additional information implicit in a given polynomial so that a specific CLA can be constructed to find the arbitrary roots of polynomials.

1) The Constrained Relation Between Roots and Coefficients of a Polynomial: It is well known that there exists the constrained relationship between the (real or complex) roots and the coefficients of an n-order polynomial as follows [27]:

$$\begin{cases} \sum_{i=1}^{n} \lambda_i = -a_1 \\ \sum_{i<j}^{n} \lambda_i \lambda_j = a_2 \\ \vdots \\ \lambda_1 \lambda_2 \cdots \lambda_n = (-1)^n a_n. \end{cases} \tag{3}$$

This is the fundamental result from polynomial theory. Therefore, if the conventional BPA can be supplemented by such a set of constrained relations to compose a CLA for finding the corresponding roots of polynomial, then the learning process will be certainly speeded up and the accuracy for the root solutions will be possibly improved.

2) The Newton Identities: In addition, another constrained relation, called as the root moment of polynomial first formulated by Isaac Newton, is introduced here, which leads to the relationships known as the Newton identities (Stathaki [26] and see the references therein). The root moment of a polynomial is defined as follows.

Definition 1: For an n-order polynomial as (1), assume that the corresponding n roots are, respectively, $\lambda_1, \lambda_2, \ldots$, and λ_n, then, the m ($m \in Z$, here Z denotes the integer domain) order root moment of the polynomial is defined as

$$S_m = \lambda_1^m + \lambda_2^m + \cdots + \lambda_n^m = \sum_{i=1}^{n} \lambda_i^m. \tag{4}$$

Obviously, S_m's are possibly complex number which depend on λ_i's. Furthermore, there hold $S_0 = n$ and $(dS_m/d\lambda_i) = m\lambda_i^{m-1}$. According to this definition of the root moment, we can readily obtain the recursive relationship between the m-order root moment and the coefficients of the polynomial as follows:

$$\begin{cases} S_1 + a_1 = 0 \\ S_2 + a_1 S_1 + 2a_2 = 0 \\ \vdots \\ S_m + a_1 S_{m-1} + \cdots + a_n S_{m-n} = 0. \end{cases} \tag{5}$$

The above recursive relationship is named as Newton identity [26]. Specifically, the above relationship can be used to calculate S_m for $m < 0$. From these Newton identities, we can obtain a theorem as follows.

Theorem 1: Suppose that an n-order polynomial as (1) is known, then, a set of parameters (root moment) $\{S_m, m = 1, 2, \ldots, n)$ is solely determined recursively via (5). Conversely, given the n root moments S_m, an n-order polynomial like (1) is solely determined recursively via (5).

For the case of $m < 0$, however, the above same conclusion can be stated as follows.

Corollary 1: Suppose that an n-order polynomial as (1) is known, then, a set of parameters (root moment) $\{S_m, m = -1, -2, \ldots, -n)$ is solely determined recursively via (5). Conversely, given the n root moments $\{S_m, m = -1, -2, \ldots, -n)$, an n-order polynomial like (1) is solely determined recursively via (5).

After discussing the constrained conditions implicit in polynomials, in Section III, we shall develop the neural network root finder model and construct a suitable complex version of the CLA for finding the arbitrary roots of arbitrary polynomials.

III. Multilayer Perceptron Network Root-Finder Model and Adaptive Complex Constrained Learning Algorithms

A. Multilayer Perceptron Network Root-Finder Model

Considering an n-order arbitrary polynomial $f(z)$, the idea for finding the complex roots of the polynomial by neural networks is to use a two-layered perceptron network to represent or approximate this polynomial so that those weights of the input-hidden layer of the network try to represent, or approximate, the complex roots of the polynomial. Assume that the two-layered perceptron network model, as shown in Fig. 1, is configured to perform the task. Obviously, this network is of the size of $2-n-1$ with a structure of two input neurons, n hidden neurons, and one output neuron. As a result, this model, which is somewhat similar to the Σ-Π neural network [28], is in essence a one-layered linear network extended by a difference-product unit. The network has two input neurons corresponding to the terms of 1 and z, and n hidden neurons of forming the difference between the input z and weights w_i's, and one output product neuron of performing the multiplications on the outputs of n hidden neurons. Only the weights between the input neuron clamped at value 1 and the hidden neurons need to be trained. The weights between the input z and the hidden neurons and those between the hidden neurons and the output neuron fixed at 1s.

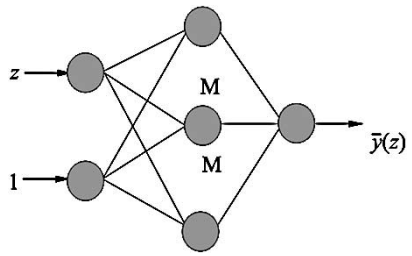


Fig. 1. A two-layered MLPN model architecture for finding complex roots of arbitrary polynomials.

From the above analyses, in mathematical terminology, the output of the ith hidden neuron in the network can be written as $\hat{u}_i = z - w_i \cdot 1 = z - w_i$, where $w_i (i = 1, 2, \ldots, n)$ are the network weights of input-to-hidden layer, i.e., the complex roots of the polynomial to be estimated. The output of the output layer performing multiplication on the outputs of the hidden layer can be denoted as

$$\hat{y}(z) = \prod_{i=1}^{n} \hat{u}_i = \prod_{i=1}^{n} (z - w_i). \tag{6}$$

The outer-supervised (target) signal setup at the output of this network model is $f(z)$. In fact, if (6) is made the logarithmic transformation after absolute operation, we can obtain the following:

$$\bar{y} = \ln |\hat{y}(z)| = \sum_{i=1}^{n} \ln |z - w_i|. \tag{7}$$

As a result, another neural root-finder model for finding the complex roots of polynomial, which is structurally similar to the factorization network model proposed by Perantonis *et al.* [16], can be easily derived. In this model, the hidden neurons, which compute the linear differences as in the above model, become nonlinear hidden neurons with a logarithmic activation function and the output neuron performs a linear summation instead of multiplication. As pointed out in [16], this structure with logarithmic activation functions has been preferred to the $(\Sigma-\Pi)$ architecture employed in [17] and [28] since it produces smoother cost function landscape avoiding deep valleys and thus facilitates learning. In the sequel, we shall focus on discussing this model.

In the following subsection, we shall present and discuss the complex CLA, which incorporates into the training of the MLPN the *a priori* knowledge about the nature of the problem in the form of constraints that is suitable for the above two neural network models for finding polynomial roots.

B. Adaptive Complex Constrained Learning Algorithms for MLPNs

It is well known that the commonly used BPA, with a gradient descent type [30], may often lead to unsatisfactory solutions of the given problem [14]–[16]. Because the error cost function defined at the output of the MLPN may include many long narrow troughs that are flat in one direction and steep in surrounding directions, the corresponding training speed is very slow [14], [15], [31], [32]. If the gradient descent is supplemented with a momentum acceleration, zigzag paths will probably be avoided

and the learning process will be accelerated [14], [15]. However, such an improvement on the BPA can meet only one target— improvement of the convergent speed. It is still difficult to realize the other two targets at the same time—better generalization capability and a smaller number of local minima. The reason is that this BPA does not take into account any additional information about the nature of the problem, except for the network architecture and the desired input-target output relation [14]–[16].

In 1995, Perantonis and Karras proposed a new CLA [14], [15] by incorporating additional information about the desired behavior of the hidden units and synaptic weights, which prove to be superior to other well-known supervised learning algorithms because of the smaller number of local minima, higher learning speed, and better generalization capability.

In this paper, we apply this CLA incorporated into the relation between roots (weights) and coefficients of polynomials to finding the complex roots of polynomials. Suppose that P training patterns are selected from the region $|z| < 1$ of complex domain C, an error cost function (ECF) can be defined at the output of the network

$$E(w) = \frac{1}{2P}\sum_{p=1}^{P}|e_p(w)|^2 = \frac{1}{2P}\sum_{p=1}^{P}(o_p - y_p)(o_p - y_p)^* \tag{8}$$

where w is the set of all weights in the neural network and the superscript $*$ denotes complex conjugate. The target signal is $o_p = \ln|f(z_p)|$ and the actual output of the network can be expressed as $y_p = \sum_{i=1}^{n}\ln|z_p - w_i|$.

The above variables z_p, w_i, and function $f(z_p)$ are all possibly complex-valued ones, so we must in the following do the derivations according to the rule of complex variables. For an arbitrary complex variable $z = x + iy$, where x and y are the real and imaginary parts of z, and i denotes $\sqrt{-1}$. For the convenience of deducing the learning algorithm, here we give a definition of the derivative of a real-valued function (e.g., the ECF) with respect to complex variable.

Definition 2: Assume a real-valued function $U(w)$ is the function of complex variable w with the real and imaginary parts, w_1 and w_2, the derivative of $U(w)$ w.r.t w is denoted as $(\partial U(w)/\partial w) = (\partial U(w)/\partial w_1) + i(\partial U(w)/\partial w_2)$.

1) Gradient Descent Algorithm: The BPA is based on the gradient descent rule, which can easily be deduced from the ECF. By taking the partial derivative of $E(w)$ with respect to (w.r.t.) w_i, we have

$$J_i = \frac{\partial E(w)}{\partial w_i} = \frac{1}{2P}\sum_{p=1}^{P}\frac{\partial E(w)}{\partial y_p}\cdot\frac{\partial y_p}{\partial w_i} = \frac{1}{P}\sum_{p=1}^{P}e_p(w)\cdot\frac{z_p - w_i}{|z_p - w_i|^2}. \tag{9}$$

As a result, the gradient descent based BPA can be described as follows:

$$dw_i = -\eta J_i \tag{10}$$

where $dw_i = w_i(k) - w_i(k-1)$ denotes the difference between $w_i(k)$ and $w_i(k-1)$, the current, and the past weight.

2) Complex Constrained Learning Algorithm: We can see in Huang [24] that the BPA has a very slow learning speed unless the corresponding suitable and effective initialized weights are given. Although the approach for initializing weights can be derived by separating different complex roots by means of the polynomial theory, it will take a longer time to achieve the accurate upper and lower bounds, especially for those higher order polynomials.

To alleviate this difficulty, we incorporate the *a priori* information such as the relationship between the complex roots, i.e., the weights and the coefficients of a polynomial, and the complex root moments of the polynomial into the gradient descent based BPA. This facilitates the learning process that leads to better solutions. Following the idea of the CLA proposed by Perantonis and Karras [14], [15], we develop a complex CLA for finding the complex roots of a polynomial. Here, the additional information available is the constraint relations defined in (3) or (5). We can uniformly write them as $\Phi = [\Phi_1, \Phi_2, \ldots, \Phi_m]^T = 0 (m \le n)$ (T denotes the transpose of a vector or matrix), which is composed of the constraint conditions of (3) or (5). Therefore, the objective of the learning process is to reach a minimum of the ECF of (8), which fulfills as good as possible the additional constraints $\Phi = 0$.

In fact, the CLA proposed by Perantonis, *et al.* [14]–[16] is essentially similar to the method proposed by Bryson and Denham [32]. In the algorithm, to avoid missing the global minimum, a constraint for updated weights is imposed. Consequently, the sum of absolute valued square of the individual weight changes takes a predetermined positive value $(\delta P)^2$

$$\sum_{i=1}^{n}|dw_i|^2 = (\delta P)^2 \tag{11}$$

where dw_i denotes the change of weight $w_i, \delta P$ is a constant. This means that, at each epoch, the search for an optimum new point in the weight space is restricted to a small hypersphere of radius δP centered at the point defined by the current weight vector. If δP is small enough, the changes to $E(w)$ and Φ induced by changes in the weights can be approximated by the first differentials $dE(w)$ and $d\Phi$.

In the following, we will derive the complex CLA based on the constraint conditions of $\Phi = 0$ and (11). First, assume $d\Phi$ to be equal to a predetermined vector quantity δQ, designed to bring Φ closer to its target (zero). To incorporate the two constraint conditions of $\Phi = 0$ and (11) into the BPA, we introduce suitable Lagrange multipliers for them so that a new built target cost function includes the effects of these two constraint conditions. Assume that a Lagrange multiplier vector $V = [\nu_1, \nu_2, \ldots, \nu_m]^T$ is needed to take into account the constraints in $\Phi = 0$ and another Lagrange multiplier μ is introduced for (11). The objective of learning process is to ensure that the maximum possible change in $|dE(w)|$ is achieved at each epoch. By introducing the function $\varepsilon, d\varepsilon$ is defined as follows:

$$d\varepsilon = dE(w) + (\delta Q^H - d\Phi^H)V + \mu\left[(\delta P)^2 - \sum_{i=1}^{n}|dw_i|^2\right] \tag{12}$$

where the superscript H denotes the conjugate transpose of a vector or matrix. As a result, we can derive

$$dw_i = \frac{J_i}{2\mu} - \frac{F_i^H V}{2\mu} \tag{13}$$

$$\mu = -\frac{1}{2}\left[\frac{I_{JJ} - I_{JF}^H I_{FF}^{-1} I_{JF}}{(\delta P)^2 - \delta Q^H I_{FF}^{-1}\delta Q}\right]^{1/2} \tag{14}$$

$$V = -2\mu I_{FF}^{-1}\delta Q + I_{FF}^{-1} I_{JF} \tag{15}$$

where $F_i = [F_i^{(1)}, F_i^{(2)}, \ldots, F_i^{(m)}]^T$, $F_i^{(j)} = (\partial\Phi_j/\partial w_i)(i = 1,2,\ldots,n, j = 1,2,\ldots,m)$, $I_{JJ} = \sum_{i=1}^{n} |J_i|^2$ is a scalar and I_{JF} is a vector whose components are defined by $I_{JF}^{(j)} = \sum_{i=1}^{n} J_i F_i^{(j)}$ $(j = 1,2,\ldots,m)$. Specifically, I_{FF} is a matrix, whose elements are defined by $I_{FF}^{jk} = \sum_{i=1}^{n} F_i^{(j)} F_i^{(k)^*}$ $(j,k = 1,2,\ldots,m)$.

Note that the derivation of the Lagrange multiplier parameters μ and V can refer to Appendix A.

Equation (13) forms a new weight update rule, called as complex CLA for the MLPN instead of the conventional BPA as described in (10).

From (14), we can see that there are $(m+1)$ parameters $\delta P, \delta Q_j (j = 1,2,\ldots,m)$, which need to be determined before the leaning process begins. Parameter δP is often selected as a fixed value. However, the vector parameters $\delta Q_j (j = 1,2,\ldots,m)$ are generally selected as proportional to Φ_j, i.e., $\delta Q_j = -k\Phi_j$ $(j = 1,2,\ldots,m$ and $k > 0)$, which ensures that the constraints Φ move toward zero at an exponential rate as the training progresses [14]–[16]. From (14), we note that k should satisfy $k \le \delta P(\Phi^H I_{FF}^{-1}\Phi)^{-1/2}$. In practice, the simplest choice for k is $k = \eta\delta P/\sqrt{\Phi^H I_{FF}^{-1}\Phi}$, where $0 < \eta < 1$ is another free parameter of the algorithm apart from δP.

Comments: It can be found that the first term of the right hand of (13) is related to the gradient $J_i = \partial E(w)/\partial w_i$, i.e., the BPA. The difference is that the former coefficient is $1/2\mu$ (which is related to the *a priori* information of polynomial), while the latter one is $-\eta$. The specific key point is that the second term of the CLA, $-F_i^H V/2\mu$ (which is also related to the *a priori* information from the problem), is *greatly helpful* to instruct the search along the global minimum of the error surface so that the optimal solutions (global minimum solutions) can be readily reached within a shorter time. Therefore, this CLA is an important learning algorithm that can ensure those neural computation problems to be exactly solved within a reasonable time.

IV. Evaluations and Discussions on Complex Constrained Learning Algorithms

A. Computational Complexity Estimates

In literature [25], we can see that the root-finding method based on this CLA is apparently faster than that based on the simple BPA [30]. However, because we have to compute at each epoch, the values of the constraint conditions of (3) or (5) used for Φ and their derivatives $\partial\Phi/\partial w$, which are sharply dominant in the entire computations of the CLA. Therefore, as the order n of polynomial $f(z)$ increases, the training time will significantly increase. In fact, we can estimate the number of multiplication operations at each epoch for using the complex CLA based on the constrained relation of (3) to find the complex roots of a given n-order polynomial $f(z)$ as the following remark [25].

Remark 1: At each epoch for finding all the complex roots of a given n-order arbitrary polynomial $f(z)$ based on the constrained learning MLPN using the constraint conditions from the first one to the mth of (3), the estimate of the number of multiplication operations needed for computing Φ and $\partial\Phi/\partial w, \mathrm{CE}_\lambda(n,m)$ is

$$\mathrm{CE}_\lambda(n,m) = 4\left(\sum_{j=0}^{m} j^2 C_n^j - \sum_{j=0}^{m} j C_n^j - \sum_{j=0}^{m} C_n^j + n + 1\right). \tag{16}$$

Specifically, when $m = n$, (16) becomes into $\mathrm{CE}_\lambda(n,n) = 4[(n^2 - n - 4)2^{n-2} + n + 1]$. Obviously, $\mathrm{CE}_\lambda(n,n)$ is of the order $O(2^n)$ multiplication operations.

Likewise, we can give the estimate of the number of multiplication operations at each epoch for computing the constraint conditions of (5) for Φ and their derivatives $\partial\Phi/\partial w$, which is stated in the following Remark 2.

Remark 2: At each epoch for finding all the complex roots of a given n-order arbitrary polynomial $f(z)$ based on the constrained learning MLPN using the constraint conditions from the first one to the mth of (5), the estimate of the number of multiplication operations needed for computing Φ and $\partial\Phi/\partial w, \mathrm{CE}_S(n,m)$ is

$$\mathrm{CE}_S(n,m) = \frac{2}{3}(m-1)(nm^2 + 10nm + 3m - 12n + 6). \tag{17}$$

Obviously, $\mathrm{CE}_S(n,m)$ is of the order $O(nm^3)$ multiplication operations, which is considerably lower than that of the original root-coefficients relation method (RCRM). Specifically, when $m = n$, (17) results in $\mathrm{CE}_S(n,n) = (2/3)(n-1)(n^3 + 10n^2 - 9n + 6)$. The deduction of this Remark 2 is stated in Appendix B.

It can be found that the ratio r_c of $\mathrm{CE}_\lambda(n,n)$ and $\mathrm{CE}_S(n,n)$ will tend to ∞ with $n \to \infty$, i.e.,

$$\lim_{n\to\infty} r_c = \lim_{n\to\infty} \frac{\mathrm{CE}_\lambda(n,n)}{\mathrm{CE}_S(n,n)} = \infty. \tag{18}$$

B. Recursive Root-Moment Method

In fact, owing to the particular form of (5), we find that in computing λ_i^m $(i = 1,2,\ldots,n)$, it is not necessary for us to visit from λ_i^1 to λ_i^m considering the relation $\lambda_i^m = \lambda_i^{m-j} \cdot \lambda_i^j$. Therefore, in computing λ_i^m we can use the result in computing λ_i^{m-1} (i.e., the stored λ_i^{m-1}), which can be performed recursively from $i = 1$ to $i = n$. Consequently, we can make a further reduction on the computational complexity.

Suppose $\mathrm{s}^{(m)}$ to denote a vector $\mathrm{s}^{(m)} = [\lambda_1^m, \lambda_2^m, \ldots, \lambda_n^m]^T$, where $\mathrm{s}^{(0)} = [1,1,\ldots,1]^T$ is a vector with all elements for 1's. Obviously, there is $\mathrm{s}^{(0)^T}\mathrm{s}^{(0)} = S_0 = n$. For the sake of convenience, in the following, we give a definition about the concept, a direct product between two vectors.

Definition 3: Assume two vectors are denoted as $\mathrm{a} = [a_1, a_2, \ldots, a_n]^T$ and $\mathrm{b} = [b_1, b_2, \ldots, b_n]^T$; then their direct product is defined as $\mathrm{a} \otimes \mathrm{b} = [a_1 b_1, a_2 b_2, \ldots, a_n b_n]^T$.

TABLE I
THE COMPARISONS OF COMPUTATIONAL COMPLEXITIES AT EACH EPOCH OF THE RCRM AND THE RMM AS WELL AS THE RRMM

n	1	2	3	4	5	6	7	8	9	10	11	12
$CE_\lambda(n,n)$	0	4	32	148	536	1692	4896	13348	34856	88108	217136	524340
$CE_S(n,n)$	0	24	128	388	896	1760	3104	5068	7808	11496	16320	22484
$CE_R(n,n)$	0	24	92	228	456	800	1284	1932	2768	3816	5100	6644

According to this definition of direct product, we can obtain the recursive formula of computing the root moments S_m as follows:

$$\mathrm{S}^{(1)} = [\lambda_1, \lambda_2, \ldots, \lambda_n]^T \tag{19}$$

$$\mathrm{S}^{(m+1)} = \mathrm{S}^{(m)} \otimes \mathrm{S}^{(1)} \tag{20}$$

$$S_{m+1} = \mathrm{S}^{(m+1)^T} \mathrm{S}^{(0)}. \tag{21}$$

As a result, we can obtain from the above recursive formula, two sequences $\{\mathrm{S}^{(1)}, \mathrm{S}^{(2)}, \ldots, \mathrm{S}^{(n)}\}$ and $\{S_1, S_2, \ldots, S_n\}$, both of which will take a total of $n(n-1)$ multiplication operations at each epoch. If we compute first their values at each epoch and store them in memory so that the following CLA can call these values, then a large number of multiplication operations will be saved. Therefore, according to these recursive computation formula, we can deduce another remark as follows.

Remark 3: At each epoch for finding all the complex roots of a given n-order polynomial $f(z)$ based on the constrained learning MLPN using the constraint conditions from the first one to the mth of (5) and (19)–(21), the estimate of the number of multiplication operations needed for recursively computing Φ and $\partial\Phi/\partial w$, $\mathrm{CE}_R(n,m)$ is

$$\mathrm{CE}_R(n,m) = 2(m-1)(2nm+m+2). \tag{22}$$

Obviously, $\mathrm{CE}_R(n,m)$ is only of the order $O(nm^2)$ multiplication operations, which is also lower than that of the foregoing root moment method (RMM). Specifically, when $m=n$, (22) becomes into $\mathrm{CE}_R(n,n) = 2(n-1)(2n^2+n+2)$. The deduction of this Remark 3 is given in Appendix B.

We can find that the ratio, r_s, of $\mathrm{CE}_S(n,n)$ and $\mathrm{CE}_R(n,n)$, will tend to ∞ with $n \to \infty$, i.e.,

$$\lim_{n\to\infty} r_s = \lim_{n\to\infty} \frac{\mathrm{CE}_S(n,n)}{\mathrm{CE}_R(n,n)} = \infty. \tag{23}$$

Table I gives the comparisons among the computational complexities (CC) at each epoch of the original RCRM, the RMM, and the recursive root-moment method (RRMM) versus the polynomial order n under the condition of all the n constraint conditions being used.[1] Obviously, from the results it is easily found that the RRMM is indeed of the lowest computational complexity as n increases. However, when n is chosen as a smaller value, the conclusion is contrary. Specifically, the computational complexity of the RCRM is lower than that of the RRM and that of the RRMM when $n < 7$ and $n < 5$, respectively, while the computational complexity of the RRMM is always lower than that of the RMM for all n values (the exceptional case is $n=2$ when their computational complexities are completely equal, i.e., 24).

[1]Moreover, the most general case for complex computations, which are four times the real computations, is considered.

C. Initial Weight Selection

As pointed out in Section III, the CLA imposes the *a priori* information into the ECF, which is different from the conventional BPA. Consequently, it is easier to reach the global minimum on the error surface than the BPA. Hence, we can choose randomly the initial weights from the uniform distribution in interval $[-1,1]$. However, sometimes, we wish to get some insights into the initial weights so that the selected weights are capable of assuring the more rapid training to be achieved. In the following, we will give two lemmas that define the upper and lower bounds of the initial roots (weights) of polynomials [27].

Lemma 1: For an n-order arbitrary polynomial $f(z)$ as (1), suppose ζ to be its arbitrary root, then ζ satisfies $|\zeta| \le M$, where $M = \max\{|a_n|, 1+|a_1|, \ldots, 1+|a_{n-1}|\}$.

This lemma gives the upper bound of all roots (weights) of polynomial $f(z)$.

Lemma 2: For an n order arbitrary polynomial $f(z)$ as (1), suppose ζ to be its arbitrary root, then ζ satisfies $|\zeta| \ge m$, where

$$m = \left(\max\left\{ \frac{1}{|a_n|}, 1 + \frac{|a_1|}{|a_n|}, \ldots, 1 + \frac{|a_{n-1}|}{|a_n|} \right\} \right)^{-1}.$$

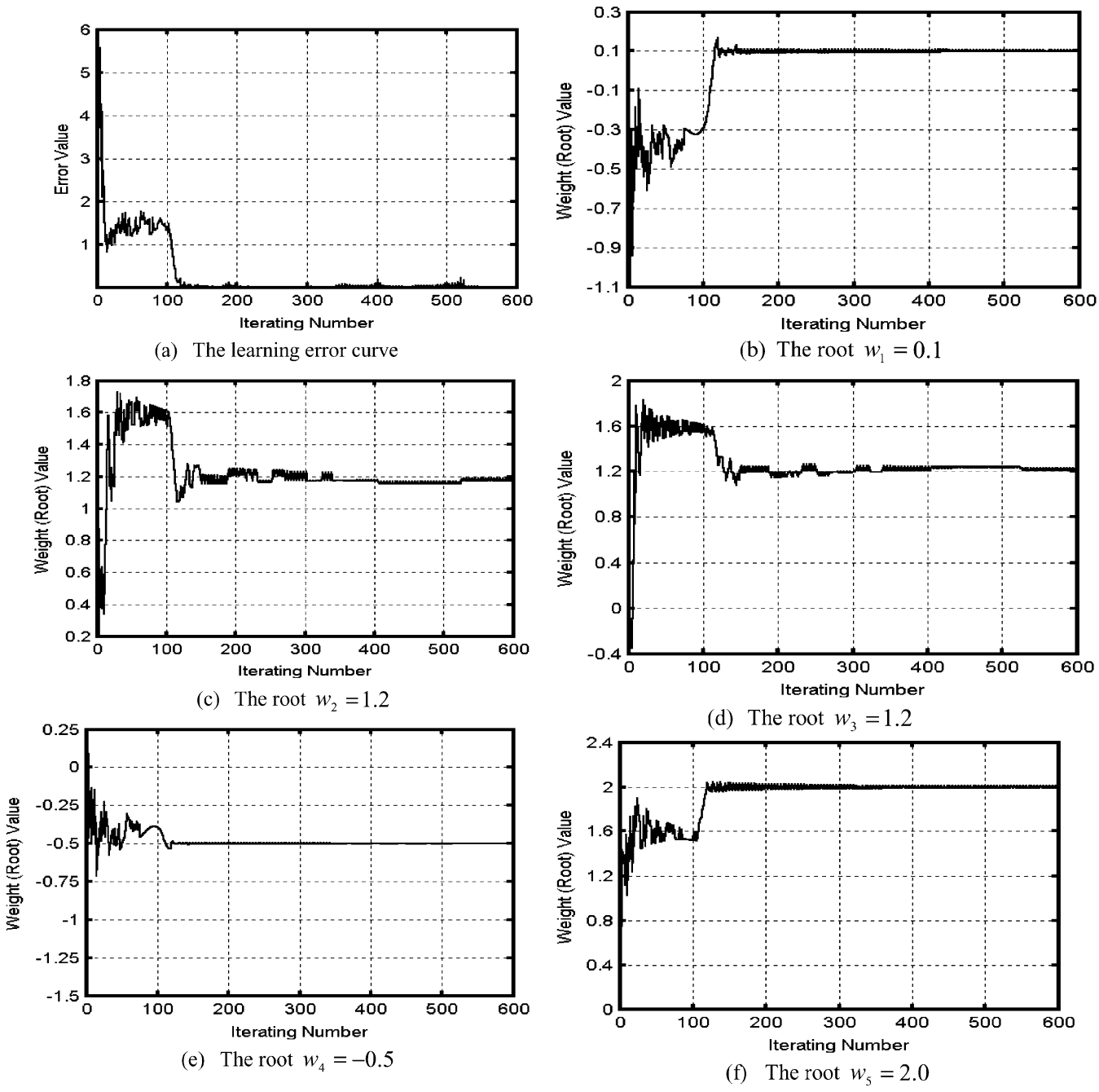
This lemma gives the lower bound of all roots (weights) of polynomial $f(z)$.

These two lemmas expound the selection method of the upper and lower bounds of the initial weights of an arbitrary polynomial. Specifically, a key point to be stressed is that this IWSM is adopted only if the consumed time for selecting the initial weights can be neglected with respect to the one for training the neural root finder involved.

D. Adaptive Learning Parameter (ALP) With the Complex CLA

From the complex CLA as described in Section III, we can see that there exists one key parameter, δP, in iterating process, which needs to be determined beforehand. If this parameter is chosen as a constant in the experiments, the training process will certainly become very slow. In fact, this parameter should become smaller and smaller as the training process progresses since the global minimum is of needle shape on the error surface. Therefore, this parameter should be initially chosen as a larger value, then it would become smaller and smaller as the training process goes. As a result, an adaptive learning parameter (ALP) scheme is proposed

$$\delta P(t) = \delta P_0 \left(1 - e^{-\frac{\theta_p}{t}}\right), \quad \delta P(0) = \delta P_0 \tag{24}$$



(a) The learning error curve

(b) The root $w_1 = 0.1$

(c) The root $w_2 = 1.2$

(d) The root $w_3 = 1.2$

(e) The root $w_4 = -0.5$

(f) The root $w_5 = 2.0$

Fig. 2. The learning error and weights (roots) curves for the root finder of the RCRM for the polynomial $f_1(z)$.

where δP_0 is the initial value for δP, which is usually chosen as a larger value; $t > 0$ is the time index for training and θ_p the scale coefficient of time t, which is usually set as $\theta_p > 1$.

V. Computer Simulations and Discussions

Before presenting the experimental results, three assumptions are made: (1) the input sample pairs $(z, f(z))$ for each polynomial involved $f(z)$ are obtained from the domain of $|z| < 1$; (2) the total number P of input training sample pairs for each polynomial involved $f(z)$ is fixed at 100; and (3) the initial weight (root) values of the MLPN, unless otherwise stated, are randomly selected from the uniform distribution in $[-1, 1]$. This section presents several examples by computer simulations to illustrate the effectiveness and efficiency of our proposed approaches. To evaluate a root finder performance (including speed and accuracy), we define two evaluation criteria as follows.

1) *The roots of a polynomial to be determined were known.* To evaluate the performance of a root finder in this case, the definition of the following average relative error criterion is used [25]:

$$d_r = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{\lambda_i - \bar{\lambda}_i}{\lambda_i} \right| \tag{25}$$

where $\bar{\lambda}_i$ is the ith complex root value of the given polynomial computed by the root finder, i.e., the weight, w_i,

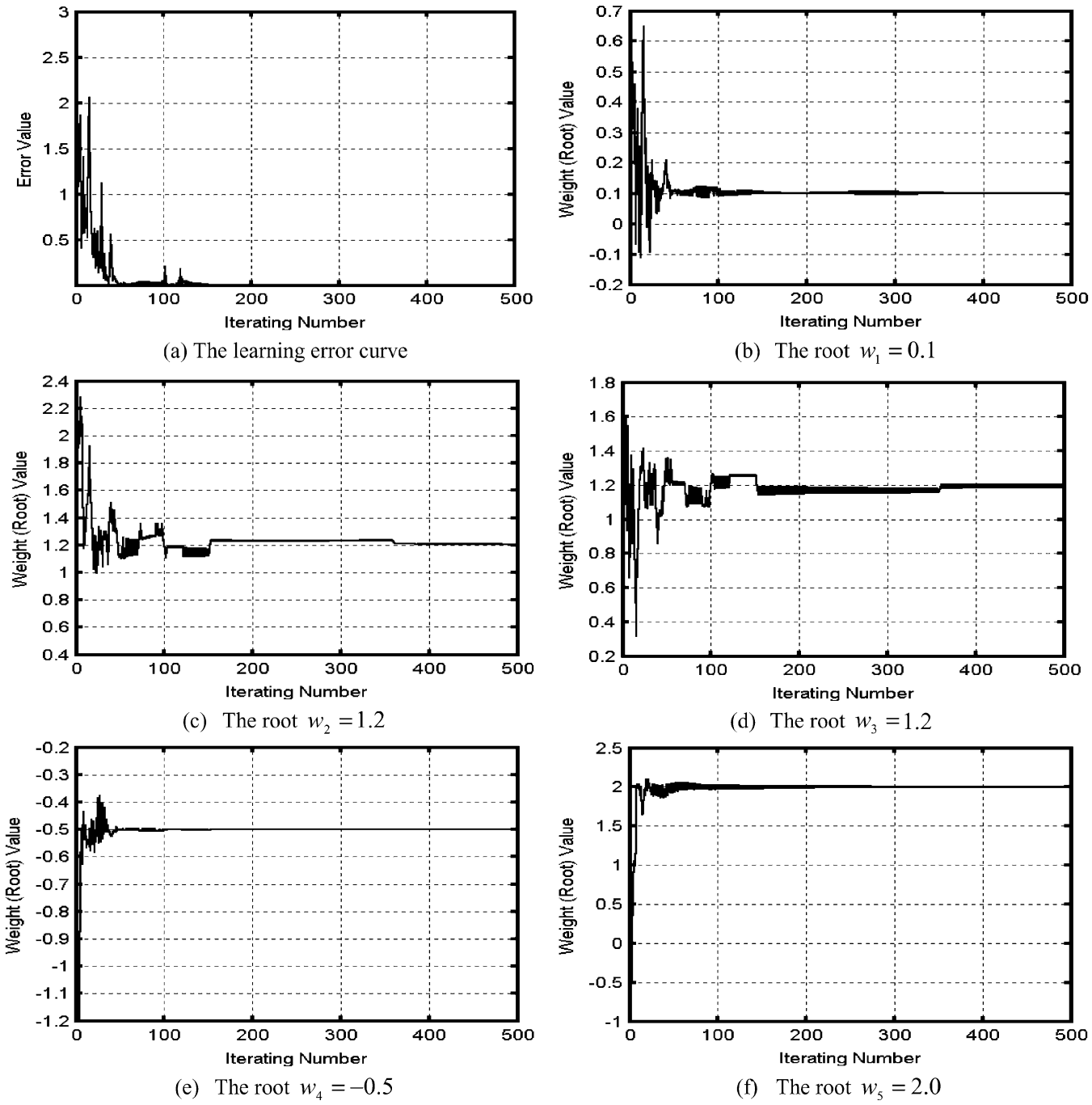


Fig. 3. The learning error and weights (roots) curves for the root finder of the RMM for the polynomial $f_1(z)$.

and λ_i is the ith true (exact) root value corresponding to this polynomial. Obviously, the better the root finder is, the smaller the index d_r is.

2) *The roots of a polynomial to be determined were unknown.* In practice, for most real problems the exact root values of the polynomials are unknown. In such a case, to evaluate the performance of a root finder, we can use another measure for the accuracy of the approximation [25]

$$C_p = \frac{1}{n}\sum_{i=1}^{n} |a_i - \bar{a}_i|^2 \tag{26}$$

where $\{a_i\}$ are the coefficients of the original polynomial $f(z)$, and $\{\bar{a}_i\}$ the coefficients of the reconstructed polynomial $\bar{f}(z)$ by the estimated roots $\{w_i\}$. Obviously, the better the root finder is, the smaller the index C_p is.

A. Finding the Roots of a Polynomial With Known Roots

In this case, assuming a fifth-order polynomial $f_1(z) = z^5 - 4z^4 + 4.43z^3 - 0.164z^2 - 1.464z + 0.144$ with the known root values $\lambda_1 = 0.1, \lambda_2 = \lambda_3 = 1.2, \lambda_4 = -0.5$ and $\lambda_5 = 2.0$, we now use an MLPN trained by the CLAs, respectively, based on the RCRM, the RMM, and the RRMM to find the corresponding

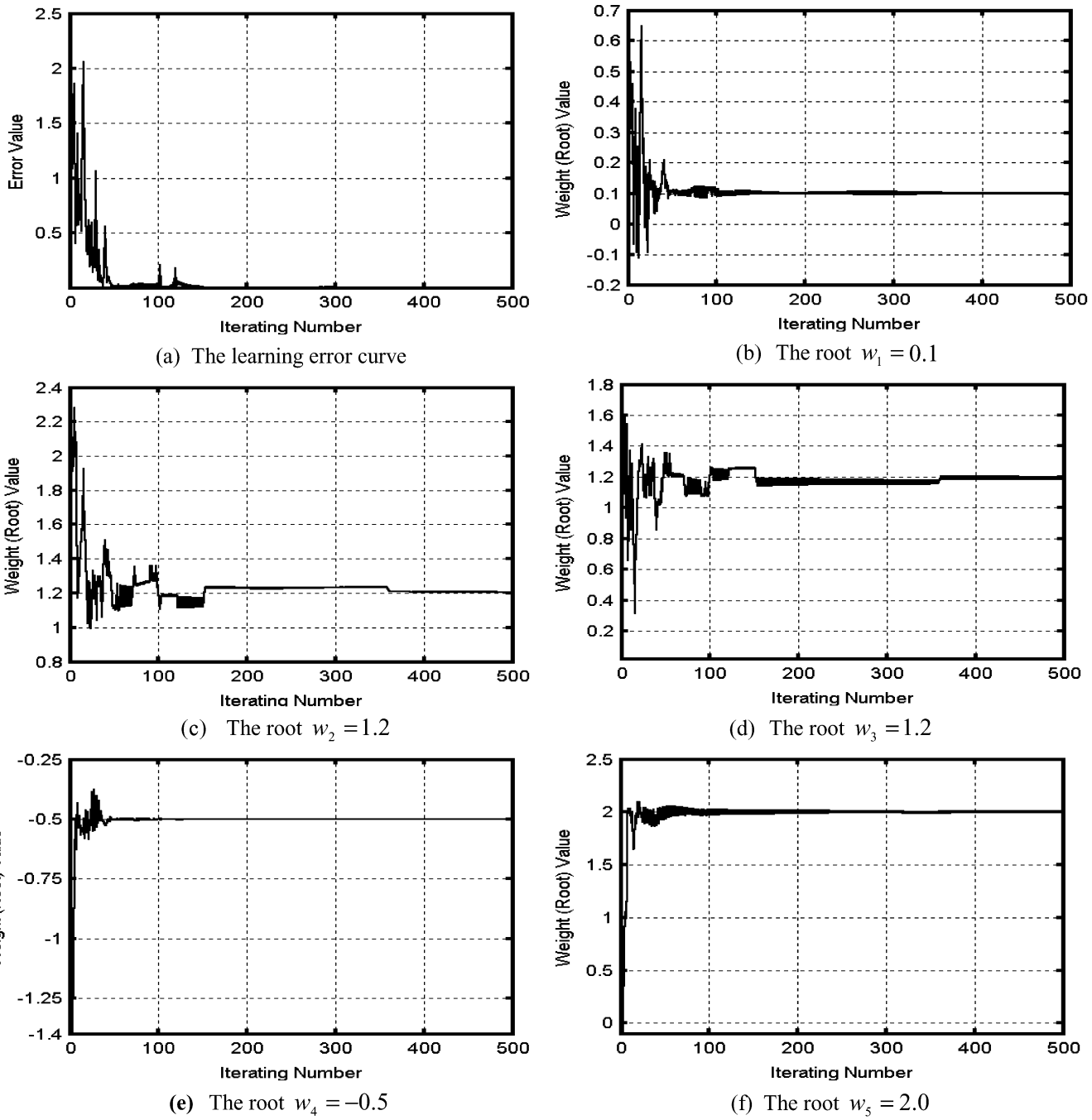


Fig. 4. The learning error and weights (roots) curves for the root finder of the RRMM for the polynomial $f_1(z)$.

roots. Assuming the termination error (TE) to be 1.0×10^{-11}, the numbers of the constrained conditions for the three methods to be $m = 4$ and without using the ALP method, it was found from the experiments that the network training would be extremely slow — even oscillate or diverge. Therefore, to speed up the training process, the ALP method for δP in (24) is adopted. For the three methods, supposing three parameters in the CLAs to be chosen as the same values, i.e., $\delta P_0 = 1.0, \theta_p = 10.0$ and $\eta = 0.75$, it could be found that the corresponding MLPNs converge to the given TE after 1076781, 74002 and 71 526 iterations, respectively, and the corresponding learning (error and weight) curves are shown in Figs. 2–4, where only the starting 500 iterations are shown. From Figs. 3 and 4, it can be seen that at the initial stage of 500 iterations, the corresponding learning (error and weight) curves for the RMM and the RRMM are almost completely identical. The reason for this result is that the RRMM is a fast recursive form of the RMM. Hence, the change traces for their corresponding errors and weight values are somewhat similar as well.

In addition, in all experiments for neural and nonneural methods we adopt Pentium III with CPU clock of 795 Mhz and RAM of 256 Mb, and use Visual Fortran 90 to encode. Consequently, we can obtain the estimated root values, the iterating numbers (IN), and the d_r's as well as the CPU times for

TABLE II
THE PERFORMANCE COMPARISONS OF THE THREE NEURAL ROOT FINDERS OF THE RCRM AND THE RMM AND THE RRMM AND THE THREE NON NEURAL ROOT FINDERS OF LAGUERRE AND NEWTON–RAPHSON AND JENKINS–TRAUB FOR THE POLYNOMIAL $f_1(z)$.

Index	w_1	w_2	w_3	w_4	w_5	IN	CPU Time (Seconds)	d_r
RCRM	0.09999987	1.2005882	1.1994117	-0.5000001	2.0000001	1076781	146.83	4.964E-05
RMM	0.1000001	1.2001810	1.1998166	-0.5000001	2.0000024	74002	21.34	3.310E-05
RRMM	0.1000001	1.2002169	1.1998809	-0.5000001	2.0000021	71526	7.21	6.115E-06
Laguerre	0.0999973	1.2005921	1.1995214	-0.5000003	2.0000004	12392	121.24	2.376E-04
Newton-Raphson	0.09999988	1.2003105	1.1996134	-0.5000002	2.0000011	123247	156.35	5.328E-05
Jenkins-Traub	0.1000002	1.2001328	1.1997345	-0.5000001	2.0000018	152321	172.92	3.737E-05

the three neural methods and three nonneural methods such as Laguerre, Newton–Raphson and Jenkins–Traub, respectively, as shown in Table II. It can be seen from Table II that the CPU time for the RCRM is all considerably longer than those for the other two neural methods, but slightly shorter than those for the three nonneural methods. Obviously, the experimental results from Fig. 2–4 and Table II show that the RRMM is of the fastest training speed. Furthermore, the estimated accuracy is higher than the other two neural methods and the three nonneural methods as well, where the reason might be explained to be that the shorter training time will alleviate the limited word-length effect for computers so that the accuracy for the root finder is relatively improved. Furthermore, the experimental results also show that the neural methods' performance is, whether in speed or accuracy, superior to the nonneural methods.

B. Finding the Roots of a Polynomials With Unknown Roots

We consider a sixth-order complex coefficient polynomial $f_2(z) = z^6 + (3+i)z^5 + (2+2i)z^4 - z^2 + (-3-i)z - 2 - 2i$, where the corresponding root values are unknown. We employ the MLPN trained by the CLAs, respectively, based on the RCRM, the RMM, and the RRMM to find the complex roots of this polynomial and compare their performances with the three nonneural methods mentioned above. According to the initial weight selection method of Lemmas 1 and 2, for $f_2(z)$ the initial weights should be chosen in the range of $0.472 \leq |w_i(0)| \leq 4.16$. Letting the TE be 1.0×10^{-14} and $\{\delta P_0 = 10.0, \eta = 0.01, \theta_p = 5.0\}$ in all the three neural methods, consequently, the corresponding learning error curves for three methods are shown in Fig. 5, which shows that the RRMM is of the fastest training speed. In addition, Table III demonstrates the estimated root values, INs, CPU times, and C_p's for the six methods. The experimental results again show that the RRMM is of the fastest training speed and a relatively higher estimated accuracy among the six methods and the neural methods are in performance (speed and accuracy) better than the nonneural methods.

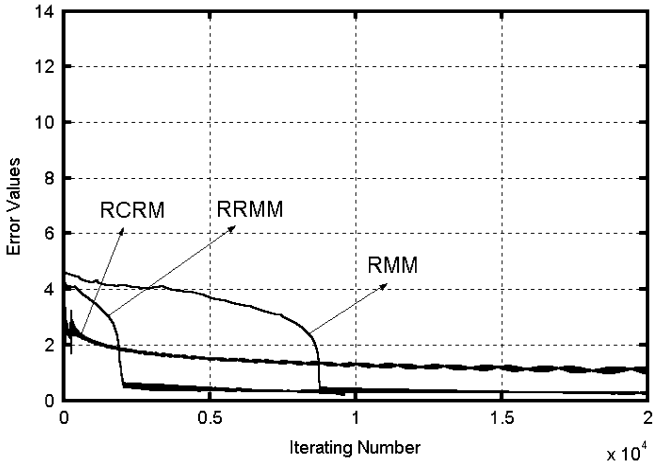


Fig. 5. The learning error curves for three methods of the RCRM and the RMM and the RRMM for finding the complex roots of the polynomial $f_2(z)$.

C. Different Number of Constrained Conditions

In this section, we investigate how the performances of the root finders are related to the number of constrained conditions. Given a four-order polynomial $f_3(z) = z^4 - 5z^3 + 4$ with known root values $\lambda_1 = -2.0, \lambda_2 = 2.0, \lambda_3 = 1.0$ and $\lambda_4 = -1.0$, we used the CLAs based on the RCRM, the RMM, and the RRMM, respectively, to train a 2-4-1 structure of MLPN for finding the real roots. Supposed four and two constrained conditions ordered from the first one of (3) or (5) to be employed, respectively, and the TE to be 1.0×10^{-10}, Table IV shows the performance comparisons between the three neural root finders with two different number of constrained conditions.

The results show that the training times of the root finders with four constrained conditions are much shorter than the training times with two constrained conditions. More experimental results show that for the cases of close roots, using the fewer constrained conditions might make the learning not converge (sometimes, oscillate in the vicinity of some points as the BPA does). It must be pointed out, however, that the accuracies of the root finders with fewer constrained conditions

TABLE III
THE PERFORMANCE COMPARISONS OF THE THREE NEURAL ROOT FINDERS OF THE RCRM AND THE RMM AND THE RRMM AND THE THREE NON NEURAL ROOT FINDERS OF LAGUERRE AND NEWTON–RAPHSON AND JENKINS–TRAUB FOR THE POLYNOMIAL $f_2(z)$

Indices	Estimated Roots	IN	CPU Time (Seconds)	C_p
RCRM	(1.1131201E-08, -0.9999999) (-1.0000002, 3.5500012E-08) (1.00000140, 1.4221039E-08) (3.3221931E-08, 0.9999999) (-0.9999996, -0.9999994) (-2.0000008 -4.4508921E-07)	1232820	236.63	2.36E-010
RMM	(-1.0500023E-08, -0.9999999) (-1.0000001, 4.5223017E-08) (0.9999999, -4.0636472E-08) (2.6403824E-08, 0.9999999) (-0.9999999, -1.0000002) (-2.0000021, -5.3523230E-08)	160689	46.54	7.45E-011
RRMM	(4.5333270E-08, -0.9999999) (-0.9999999, 7.7600233E-08) (1.0000000, -7.6832020E-09) (2.6200932E-08, 0.9999999) (-1.0000000, -1.0000001) (-2.0000000, -1.3721032E-08)	83138	13.71	8.64E-011
Laguerre	(3.7621321E-08, -0.9999997) (-1.0003101, 6.6329801E-08) (1.0000210, 5.6823034E-08) (5.2783234E-08, 0.9999998) (-0.9999994, -0.9999991) (-2.0000321, -7.5617433E-07)	92878	244.82	5.34E-09
Newton-Raphson	(2.4546367E-08, -0.9999998) (-1.0000031, 4.4396212E-08) (1.0000005, 3.9848201E-08) (2.7934292E-08, 0.9999998) (-0.9999995, -0.9999993) (-2.0000042, 3.4345902E-07)	334328	287.36	3.27E-010
Jenkins-Traub	(1.7223401E-08, -0.9999999) (-1.0000002, 5.5625401E-08) (1.0000201, 4.3240230E-08) (5.3423454E-08, 0.9999999) (-0.9999995, -0.9999996) (-2.0000020, -5.4234810E-07)	543256	313.64	2.45E-010

TABLE IV
THE PERFORMANCE COMPARISONS OF THE THREE NEURAL ROOT FINDERS OF THE RCRM AND THE RMM AND THE RRMM WITH FOUR AND TWO CONSTRAINED CONDITIONS FOR THE POLYNOMIAL $f_3(z)$

Index		Parameters	Computed root values	TIN	CPU Time (Second)	d_r
RCRM	4 constrained conditions	$\delta P_0 = 1.0$ $\theta_p = 5.0$ $\eta = 0.1$	-2.00000209 2.00000254 0.99999883 -0.99999886	19151	2.36	1.157E-06
	2 constrained conditions	$\delta P_0 = 1.0$ $\theta_p = 5.0$ $\eta = 0.1$	-2.00000011 2.00000098 0.99999873 -0.99999960	373266	7.64	5.518E-07
RMM	4 constrained conditions	$\delta P_0 = 5.0$ $\theta_p = 5.0$ $\eta = 0.1$	-2.00000352 2.00000348 0.99999988 -0.99999984	441	1.23	9.458E-07
	2 constrained conditions	$\delta P_0 = 5.0$ $\theta_p = 5.0$ $\eta = 0.1$	-2.00000085 2.00000169 0.99999871 -0.99999955	1791239	3.32	7.513E-07
RRMM	4 constrained conditions	$\delta P_0 = 5.0$ $\theta_p = 5.0$ $\eta = 0.1$	-2.00001054 2.00001071 0.99999988 -1.00000005	251	0.8 1	2.699E-06
	2 constrained conditions	$\delta P_0 = 5.0$ $\theta_p = 8.0$ $\eta = 0.1$	-2.00000103 2.00000186 0.99999871 -0.99999954	2826496	2.96	7.995E-07

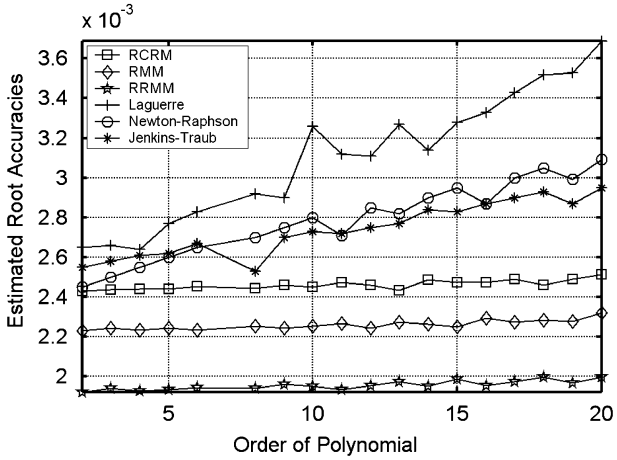


Fig. 6. The six sets of curves of the average estimated roots accuracies of the three neural root finders and the three nonneural root finders versus the order of polynomial $f_4(z)$.

are higher than those with more constrained conditions in the case of the same termination error that can be seen from Table IV since more iterating numbers are needed.

Comments: It can be found from the experimental results that the more the number of the constrained conditions is, the faster the convergent speed is. In general, when the constrained conditions are fully used to design the CLA, it will be easier for the algorithm to converge to the global minima. In addition, to avoid as the local minima as possible, it is suggested that the initial weight selection method introduced above should be adopted, especially for the case of the less number of the constrained conditions being obtained.

D. Higher Order Polynomial Case

Finally, to observe the effects of the order n of a polynomial on the estimated root accuracy, we use L-order polynomial $f_4(z)$ with known root values $r_i = \exp(j2(\pi/L)i)(i = 1, 2, \ldots, L)$ to conduct related experiments. Assuming the TE $e_r = 1.0 \times 10^{-6}$ and the controlling parameters $\{\delta P_0 = 1.0, \theta_p = 4.0, \eta = 0.7\}$ to be kept unchanged in experiments, for each root finder (including the three neural ones and the three nonneural ones), we change the order of the polynomial from $L = 2$ to $L = 20$ and repeat, respectively, 40 experiments by choosing different initial connection weights from the uniform distribution in $[-1, 1]$. Consequently, by (25), we can obtain six set of curves of the average estimated root accuracies corresponding to the six root-finding methods, as shown in Fig. 6. From Fig. 6, it shows that the neural root-finding methods are almost not sensitive to the order of the polynomial (however, the accuracies slowly decreases due to a high load of computational complexity as the order of polynomial increases), while the nonneural methods' accuracies significantly decrease as the order of the polynomial increases, especially for the Laguerre's.

Further, for an $L = 15$ order polynomial $f_5(z)$, assuming the controlling parameters are fixed as above under four different termination errors: (a) $e_r = 0.1$, (b) $e_r = 0.01$, (c) $e_r = 0.001$, and (d) $e_r = 0.0001$, we use the CLA-RRMM to repeatedly train the corresponding MLPN root finder with 40 different random initial connection weights in $[-1, 1]$ until the four TEs are reached. The estimated roots distributions in the complex plane are illustrated in Fig. 7. From Fig. 7, it can be seen that the estimated roots' accuracies become higher and higher and the estimated roots' scatters smaller and smaller as the termination accuracy increases. At the same time, these results also show that our proposed MLPN root finders are independent of the initial weight values. In other words, even if the initial roots are randomly chosen, our proposed MLPN root finders can also rapidly find the zeros of polynomials.

VI. Concluding Remarks

This paper discussed how to use MLPNs to find the roots (including real or complex) of arbitrary polynomials, which is an important research topic for neural computations and digital signal processing. The problem of finding roots of polynomials may be mapped to MLPNs similar to a function approximation problem. To speed up the training speed for the roots-finding MLPNs, the *a priori* information from the problems such as the root-coefficients relation and the root moments implicit in the polynomials must be employed to construct a new complex constrained learning algorithm (CLA). This paper discusses how to integrate these constrained conditions available from polynomials into the training of MLPNs for the finding roots. We investigated imposing the root-coefficients relation and the root moments, respectively, into the conventional BPA for formulating the corresponding CLAs. Specifically, for the *a priori* information of the root moments from polynomial, we derived a recursive root moments based CLA to train a suitable structure of MLPNs for finding the roots of polynomials, which is of more rapid training speed. Moreover, we made the estimates of computation complexities on the RCRM, the RMM, and the RRMM, respectively. It was found from both theory and experiments that the RRMM is of lowest computation complexity among the three methods. In addition, we proposed an adaptive learning parameter method for selecting the weight bound parameter δP and gave the initial weight selection method for the roots finding MLPNs. Finally, by computer simulations, it was showed that the RRMM is the best method among the three methods not only in the training speed but also in the estimated accuracy.

Specifically, in this paper, we also discussed and compared the performance of the three neural root finders based on the RCRM, the RMM, and the RRMM, respectively, and the three nonneural root finders such as Laguerre, Newton–Raphson, and Jenkins–Traub methods. The experimental results showed that although coded by Visual Fortran 90 program and performed on conventional von Neumann sequential architecture computer with an inherently parallel algorithm, the three neural root finders have significantly faster training speeds and better convergent root accuracies with respect to the three nonneural methods.

Further works will include how to find, by neural connectionism method, the roots with the maximum and minimum moduli rapidly for a given polynomial so as to exactly judge whether a polynomial can be utilized to design a minimum phase system in signal processing.

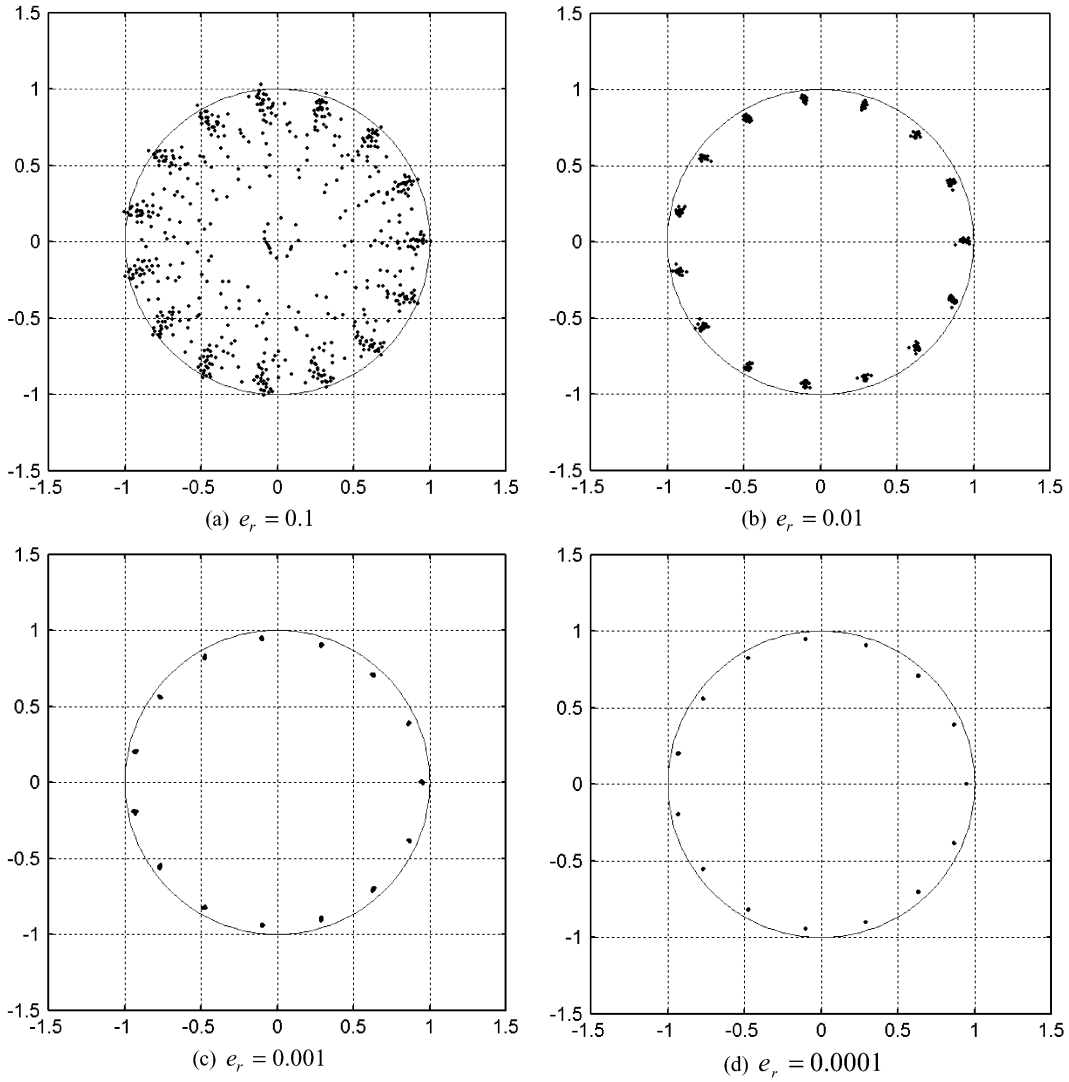


Fig. 7. The estimated roots distributions of $f_5(z)$ in the complex plane for the MLPN –RRMM root finder model under different termination errors. (a) $e_r = 0.1$. (b) $e_r = 0.01$,. (c) $e_r = 0.001$. (d) $e_r = 0.0001$.

APPENDIX A

The derivations of (14) and (15) are stated as follows.

By substituting (13) into the condition $\delta Q = \sum_{i=1}^{n} dw_i F_i$, we have $\delta Q = \sum_{i=1}^{n}(J_i F_i - F_i F_i^H V/2\mu) = (I_{JF} - I_{FF}V/2\mu)$, where $I_{JF} = \sum_{i=1}^{n} J_i F_i, I_{FF} = \sum_{i=1}^{n} F_i F_i^H$, we have $V = -2\mu I_{FF}^{-1}\delta Q + I_{FF}^{-1} I_{JF}$, which is just (15). By substituting (13) into (11), we have $\sum_{i=1}^{n}(J_i - F_i^H V)^*(J_i - F_i^H V) = 4\mu^2(\delta P)^2$, which can be further rewritten as $I_{JJ} - I_{JF}^H V - V^H I_{JF} + V^H I_{FF} V = 4\mu^2(\delta P)^2$, where $I_{JJ} = \sum_{i=1}^{n} |J_i|^2$. Considering (15) and $(I_{FF}^{-1})^H = I_{FF}^{-1}$ and $\mu < 0$, we have $\mu = -(1/2)[(I_{JJ} - I_{JF}^H I_{FF}^{-1} I_{JF}/(\delta P)^2 - \delta Q^H I_{FF}^{-1}\delta Q)]^{1/2}$, which is just (14).

APPENDIX B

Lemma B.1: The squared sum of n natural numbers is $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$.

The derivation of Remark 2 is presented as follows.

1) In the most general case of complex number, the number of the multiplication operations for computing the m constrained conditions ordered from the first one of Φ of (5) at each epoch $\mathrm{CE}_1(n,m)$ is

$$\begin{aligned}\mathrm{CE}_1(n,m) &= 4\sum_{i=1}^{m}[n(i-1) + n(i-2) + \cdots \\ &\quad + n(2-1) + n(1-1)] + 4\sum_{i=2}^{m} i \\ &= 4n\sum_{i=1}^{m}\frac{i(i-1)}{2} + 2(m+2)(m-1) \\ &= \frac{2}{3}(m-1)(nm^2 + nm + 3m + 6). \quad \text{(B.1)}\end{aligned}$$

2) In the most general case of complex number, the number of the multiplication operations for computing the m

constrained conditions ordered from the first one of the derivations $\partial\Phi/\partial w_i$ at each epoch $\mathrm{CE}_2(n,m)$ is

$$\mathrm{CE}_2(n,m) = 4\sum_{i=3}^{m} n(i-2) + 4n(m-1) + 8\sum_{i=3}^{m} n(i-2) = 2n(m-1)(3m-4). \quad \text{(B.2)}$$

We can then compute the total number of the multiplication operations for the m constrained conditions ordered from the first one of Φ and $\partial\Phi/\partial w_i$ at each epoch $\mathrm{CE}_S(n,m)$

$$\mathrm{CE}_S(n,m) = \mathrm{CE}_1(n,m) + \mathrm{CE}_2(n,m) = \frac{2}{3}(m-1)(nm^2+10nm+3m-12n+6). \quad \text{(B.3)}$$

Specifically, when $m = n$, (B.3) becomes into $\mathrm{CE}_S(n,n) = (2/3)(n-1)(n^3+10n^2-9n+6)$. Obviously, the limitation value of the ratio r_c of $\mathrm{CE}_\lambda(n,n)$ and $\mathrm{CE}_S(n,n)$ can be derived

$$\lim_{n\to\infty} r_c = \lim_{n\to\infty} \frac{\mathrm{CE}_\lambda(n,n)}{\mathrm{CE}_S(n,n)} = \lim_{n\to\infty} \frac{(n^2-n-4)2^{n-2}+n+1}{\frac{2}{3}(n-1)(n^3+10n^2-9n+6)} = \frac{3}{8}\lim_{n\to\infty}\frac{2^n}{n^2} = \infty. \quad \text{(B.4)}$$

Q.E.D.

The derivation of Remark 3 is given as follows.

1) The number of the multiplication operations for computing the m terms ordered from the first one of the two complex sequences $\{\mathrm{s}^{(1)}, \mathrm{s}^{(2)}, \ldots, \mathrm{s}^{(n)}\}$ and $\{S_1 S_2, \ldots, S_n\}$ at each epoch is $\mathrm{CE}_0(n,m) = 4n(m-1)$.
2) The number of the multiplication operations for computing the m terms ordered from the first one of the complex constraint conditions Φ of (5) apart from the above two sequences at each epoch is $\mathrm{CE}_1(n,m) = 4\sum_{i=2}^{m} i = 2(m-1)(m+2)$.
3) The number of the multiplication operations for computing the m terms ordered from the first one of $\partial\Phi/\partial w$ of (5) apart from the above two sequences at each epoch is $\mathrm{CE}_2(n,m) = 4n(m-1) + 8\sum_{i=3}^{m} n(i-2) = 4n(m-1)^2$. Consequently, we can compute the total number of the multiplication operations for computing the m terms ordered from the first one of Φ and $\partial\Phi/\partial w_i$ at each epoch $\mathrm{CE}_R(n,m) = \sum_{i=0}^{2}\mathrm{CE}_i(n,m) = 2(m-1)(2nm+m+2)$. Specifically, when $m = n$, $\mathrm{CE}_R(n,n) = 2(n-1)(2n^2+n+2)$. Moreover, the limitation value of the ratio r_s of $\mathrm{CE}_S(n,n)$ and $\mathrm{CE}_R(n,n)$ can be derived

$$\lim_{n\to\infty} r_s = \lim_{n\to\infty} \frac{\mathrm{CE}_S(n,n)}{\mathrm{CE}_R(n,n)} = \lim_{n\to\infty} \frac{\frac{2}{3}(n-1)(n^3+10n^2-9n+6)}{2(n-1)(2n^2+n+2)} = \lim_{n\to\infty}\frac{1}{6}n = \infty \quad \text{(B.5)}$$

Q.E.D.

REFERENCES

[1] S. R. Tate, "Stable computation of the complex roots of unity," *IEEE Trans. Signal Processing*, vol. 43, pp. 1709–1711, July 1995.
[2] F. Y. Yan and C. C. Ko, "Method for finding roots of quartic equation with application to RS codes," *Electron. Lett.*, vol. 34, pp. 2399–2400, 1998.
[3] T. N. Lucas, "Finding roots of polynomials by using the Routh array," *Electron. Lett.*, vol. 32, pp. 1519–1521, 1996.
[4] A. Aliphas, S. S. Narayan, and A. M. Peterson, "Finding the zeros of linear phase FIR frequency sampling filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 729–734, June 1983.
[5] C. E. Schmidt and L. R. Rabiner, "A study of techniques for finding the zeros of linear phase FIR digital filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, pp. 96–98, Feb. 1977.
[6] K. Steiglitz and B. Dickinson, "Phase unwrapping by factorization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-30, pp. 984–991, Dec. 1982.
[7] B. Thomas, F. Arani, and B. Honary, "Algorithm for speech model root location," *Electron. Lett.*, vol. 33, pp. 355–356, 1997.
[8] D. Starer and A. Nehorai, "Adaptive polynomial factorization by coefficient matching," *IEEE Trans. Signal Processing*, vol. 39, pp. 527–530, Feb. 1991.
[9] H. P. William, A. T. Saul, T. V. William, and P. F. Brian, *Numerical Recipes in Fortran*, 2nd ed. New York: Cambridge Univ. Press, 1992, ch. 9.
[10] R. Anthony and R. Philip, *A First Course in Numerical Analysis*. New York: McGraw-Hill, 1978.
[11] P. Henrici, *Elements of Numerical Analysis*. New York: Wiley, 1964.
[12] M. Lang and B. C. Frenzel, "Polynomial root finding," *IEEE Signal Processing Lett.*, vol. 1, pp. 141–143, Oct. 1994.
[13] L. Hoteit, "FFT-based fast polynomial rooting," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing 2000 (ICASSP'00)*, vol. 6, Istanbul, Turkey, June 2000, pp. 3315–3318.
[14] S. J. Perantonis and D. A. Karras, "An efficient constrained learning algorithm with momentum acceleration," *Neural Networks*, vol. 6, pp. 237–249, 1995.
[15] D. A. Karras and S. J. Perantonis, "An efficient constrained training algorithm for feedforward networks," *IEEE Trans. Neural Networks*, vol. 6, pp. 1420–1434, Nov. 1995.
[16] 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 Lett.*, vol. 7, pp. 5–14, 1998.
[17] D. S. Huang and Zheru Chi, "Neural networks with problem decomposition for finding real roots of polynomials," in *Proc. Int. Joint Conf. on Neural Networks 2001 (IJCNN'01)*, Washington, DC, July 15–19, 2001, Addendum, pp. 25–30.
[18] D. R. Hush and B. G. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Mag.*, vol. 10, pp. 8–39, Jan. 1993.
[19] D. S. Huang and S. D. Ma, "Linear and nonlinear feedforward neural network classifiers: A comprehensive understanding," *J. Intelligent Syst.*, vol. 9, pp. 1–38, 1999.
[20] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley, 1992.
[21] D. S. Huang, "One-layer linear perceptron for the inversion of nonsingular matrix," in *Proc. Internat. Conf. Robot, Optimization, Vision, Parallel Industry Automation (ROVPIA'96)*, Ipoh, Malaysia, Nov. 28–30, 1996, pp. 639–643.
[22] D. S. Huang and Z. Chi, "Solving linear simultaneous equations by constraining learning neural networks," in *Proc. Int. Joint Conf. on Neural Networks 2001 (IJCNN2001)*, Washington, DC, July 15–19, 2001, Addendum, pp. 31–26.
[23] L. Kinderman, A. Lewandowski, and P. Protzel, "A framework for solving functional equations with neural networks," in *Proc. Neural Information Processing, ICONIP'01*, vol. 2, Shanghai, China, Nov. 14–17, 2001, pp. 1075–1078.
[24] D. S. Huang, "Finding roots of polynomials based on root moments," in *Proc. Eighth Internat. Conf. on Neural Information Processing (ICONIP'01)*, vol. III, Shanghai, China, Nov. 14–18, 2001, pp. 1565–1571.
[25] 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*, Oct. 28–31, 2002, pp. 1516–1520. Tech. Rep..

[26] T. Stathaki, "Root moments: A digital signal-processing perspective," *Inst. Elect. Eng. Proc. Vision Image Signal Processing*, vol. 145, pp. 293–302, Aug. 1998.

[27] *Handbook of Mathematics*, Publ. House High Educat., Beijing, China, 1979, pp. 87–116. D. Z. Fang, in Chinese.

[28] R. Hormis, G. Antonion, and S. Mentzelopoulou, "Separation of two-dimensional polynomials via a $\Sigma - \Pi$ neural net," in *Proc. Int. Conf. Modeling and Simulation*, Pittsburg, PA, 1995, pp. 304–306.

[29] D. S. Huang, H. H. S. Ip, Z. Chi, and H. S. Wong, "Dilation method for finding close roots of polynomials based on constrained learning neural networks," *Phys. Lett. A*, vol. 309, pp. 443–451, 2003.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, ch. 8, pp. 318–362.

[31] D. R. Hush, B. Horne, and J. M. Salas, "Error surfaces for multilayer perceptrons," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 1152–1161, Sep./Oct. 1992.

[32] A. E. Bryson and W. F. Denham, "A steepest-ascent method for solving optimum programming problems," *J. Appl. Mechan.*, vol. 29, pp. 247–257, 1962.

De-Shuang Huang (SM'98) received the B.Sc. degree in electronic engineering from the Institute of Electronic Engineering, Hefei, China, in 1986, the M.Sc. degree in electronic engineering from the National Defense University of Science and Technology, Changsha, China, in 1989, and the Ph.D. degree from Xidian University, Xian, China, in 1993.

He was a Postdoctoral Student at Beijing Institute of Technology, Beijing, China, and at the National Key Laboratory of Pattern Recognition, Chinese Academy of Sciences, Beijing, China, from 1993 to 1997. In September 2000, he joined the IIM/CAS as the recipient of "Hundred Talents Program of CAS." From September 2000 to March 2001, he worked as Research Associate at the Hong Kong Polytechnic University. From April 2002 to June 2003, he worked as a Research Fellow in City University of Hong Kong. From August to September 2003, he was at George Washington University, Washington DC, as a Visiting Professor. From October to December 2003, he worked as Research Fellow at the Hong Kong Polytechnic University. He is currently an associate editor of *International Journal of Information Fusion*. He has published over 150 papers and, in 1996, he published *Systematic Theory of Neural Networks for Pattern Recognition* Printing House of Electronic Industry of China, Beijing. In 2001, he published *Intelligent Signal Processing Technique for High Resolution Radars* Printing House of Electronic Industry of China.

Dr. Huang was awarded the Second-Class Prize of the Eighth Excellent High Technology Books of China in 1996.