

Topic Category: Others (6: Neural Computation)

Finding Roots of Polynomials Based on Root Moments

De-Shuang Huang

Hefei Institute of Intelligent Machines, Chinese Academy of Sciences, Hefei, China

emails: dshuang@mail.iim.ac.cn

Abstract

This paper proposes using root moment method to find the roots of arbitrary polynomials. It has been proved that feedforward neural networks (FNN) trained with constrained learning (CL) back propagation (BP) algorithm can be used to estimate the distributions of roots of polynomials. It has been found, however, that using the different priori information implicit in the polynomials for CL-BP algorithm will result in different training time and different estimate values. By comparison, we find that the root moment method implicit in the polynomials can obviously lower the training time and leads to more accurate estimates. Therefore, in this paper the root moment method is employed to find the roots of arbitrary polynomials. This paper presents some computer simulation results, which support our claims.

Keywords: Feedforward Neural Networks, Roots, Polynomials, Root Moments, Constrained Learning.

1 Introduction

We have found that feedforward neural networks (FNN) can be used to find the roots of polynomials [1], which is an important research topic for neural computation and digital signal processing. Although there have existed many traditional methods such as iterating methods or recursive methods for finding the roots of polynomials, however, neural networks provide more flexible structure and learning algorithm for solving this problem. Nevertheless, almost all methods at present had difficulty achieving both accuracy and a high processing speed [1,2].

In [1], in order to lower the computation complexity for finding roots of polynomials, we proposed using partitioning method to recursively find arbitrary

number of roots at a time. In other words, use dilation method to progressively derive all roots. Obviously, as the order of polynomial increases, the dilation method will result in the accumulation of the errors by limited precision in computer, which will propagate the errors to the following estimate roots. This problem will become very serious for very large order of polynomials.

Focused on the roots finding method based on the prior information of the relation between roots and coefficients of polynomials, it is seen that a large number of computations are spent on computing the constrained relation between roots and coefficients of polynomials and the corresponding derivative. Therefore, if we can look for a method with fewer computations of constrained relation instead of computing the complicated relation between roots and coefficients, the training time will largely lower.

In fact, another constrained learning method based on the constrained relations between the root moments [3] and coefficients of polynomial can be employed to solve this problem. It can be proved that the corresponding computational complexity is obviously lower than the root-coefficient method's. 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 will discuss this method. The experimental results show that this method is of higher accuracy of roots than the partitioning method. Moreover, for medium order polynomial the training time for the former is also less.

This paper is organized as follows. Section 2 discusses the general principle of the neural network based approach for finding the roots of polynomials. Section 3 presents the constrained relation between the root moment and the coefficients of polynomial and the constrained

learning algorithm. Experimental results are reported and discussed in Section 4. Section 5 gives some concluding remarks.

2 The Roots Finding Neural Network Model

Assume that a n order polynomial $f(x)$ is denoted as:

$$f(x) = a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n \quad (1)$$

where $n \geq 2, a_0 \neq 0$. Without losing generality, the coefficient a_0 of x^n is usually set as 1. Suppose that there exist n approximate real roots in eqn (2), then eqn (1) can be factorized as follows:

$$f(x) = x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n \approx \prod_{i=1}^n (x - w_i) \quad (2)$$

where w_i is the i th real root of $f(x)$.

Now, the problem is how a neural network can be used to find these real roots.

The idea to design the neural network model for finding real roots of polynomials is to expand the corresponding polynomial into a FNN, i.e., use a FNN to express or approximate the polynomial. The hidden layer weights of the FNN represent the coefficients of the individual linear monomial factors such as 1 or x . As a result, a two-layer feedforward neural network model for finding real roots (FNNMFRR) of polynomials is designed (as shown in Fig. 1).

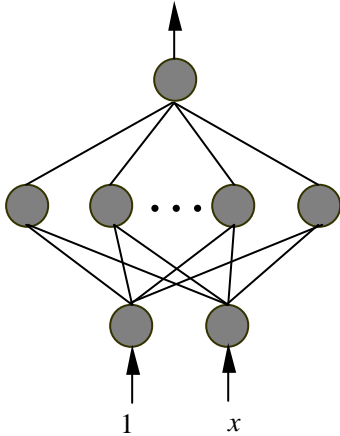


Fig.1 A two-layer feedforward neural network model for finding real roots (FNNMFRR) of polynomials

This model, which is somewhat similar to the $\Sigma - \Pi$ neural network [4], is in essence a one-layered linear network extended by a difference-product unit. The network has two input nodes corresponding to the terms of 1 and x , and n hidden nodes of forming the difference between input x and weights w_i 's and one output product node of performing the multiplications on the outputs of n hidden nodes. Only the weights between the input node clamped at value 1 and the hidden nodes need to be trained. The weights between the input ' x ' and the hidden nodes and those between the hidden nodes to the output node are fixed at 1's.

In mathematical terminology, the output corresponding to the i th hidden node can be denoted as:

$$y_i = x - w_i \cdot 1 = x - w_i \quad (3)$$

where w_i ($i=1,2,\dots,n$) are the network weights, i.e., the real roots of the polynomial.

The output of the output layer performing multiplication on the outputs of the hidden layer can be written as:

$$\hat{y}(x) = \prod_{i=1}^n y_i = \prod_{i=1}^n (x - w_i) \quad (4)$$

The outer-supervised signal defined at the output of this network model is $f(x)$.

In fact, eqn (4) can be made the logarithmic transformation, thus we can obtain the following equation:

$$\bar{y}(x) = \ln|\hat{y}(x)| = \sum_{i=1}^n \ln|x - w_i| \quad (5)$$

As a result, another neural network model for finding the real roots of a polynomial, which is structurally similar to the factorization network model proposed by Perantonis, et.al and Hormis, et.al [5,6], can be easily derived. In this case, the hidden nodes which compute the linear differences, with linear activation function in the above model become nonlinear hidden nodes with a logarithmic activation function, and the output node performs a linear summation instead of multiplication.

3 The Roots Moment and Constrained Learning Algorithm

3.1 The relations between roots and coefficients of 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 [7]:

$$\begin{cases} \sum_{i=1}^n w_i = -a_1 \\ \sum_{i<j}^n w_i w_j = a_2 \\ \vdots \\ w_1 w_2 \cdots w_n = (-1)^n a_n \end{cases} \quad (6)$$

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

In order to describe the algebraic properties of the polynomial under consideration, a concept called as the root moment of polynomials first formulated by Isaac Newton is introduced here, which leads to the relationships known as the Newton identities [3, and see the references therein]. The root moment of polynomials is defined as follows:

Definition 1: For an n order polynomial described by eqn (1), assume that the corresponding n roots are respectively w_1, w_2, \dots , and w_n , then the m ($m \in Z$) order root moment of polynomials is defined as

$$S_m = w_1^m + w_2^m + \cdots + w_n^m = \sum_{i=1}^n w_i^m \quad (7)$$

Obviously, S_m 's are possibly complex number which depends on w_i 's. Furthermore, there hold

$$S_0 = n \text{ and } \frac{dS_m}{dw_i} = m w_i^{m-1}.$$

According to this definition of the root moment, we can obtain the recursive relationship between the m order root moment and the coefficients of polynomials as follows:

(1). For the case of $m \geq 0$, we have [3]:

$$\begin{cases} S_1 + a_1 = 0 \\ S_2 + a_1 S_1 + 2a_2 = 0 \\ \vdots \\ S_m + a_1 S_{m-1} + \cdots + m a_m = 0, \quad (m \leq n) \\ S_m + a_1 S_{m-1} + \cdots + a_n S_{m-n} = 0, \quad (m > n) \end{cases} \quad (8)$$

(2). For the case of $m < 0$, we have (see the derivations of Appendix A):

$$\begin{cases} a_n S_{-1} + a_{n-1} = 0 \\ a_n S_{-2} + a_{n-1} S_{-1} + 2a_{n-2} = 0 \\ \vdots \\ a_n S_m + a_{n-1} S_{m+1} + \cdots + m a_{n+m} = 0, \quad (m \geq -n, a_0 = 1) \\ a_n S_m + a_{n-1} S_{m+1} + \cdots + S_{m+n} = 0, \quad (m < -n) \end{cases} \quad (9)$$

In fact, we can prove the equivalence of the two formulae:

$$S_m + a_1 S_{m-1} + \cdots + a_n S_{m-n} = 0, \quad (m > n) \quad (8a)$$

and

$$a_n S_m + a_{n-1} S_{m+1} + \cdots + S_{m+n} = 0, \quad (m < -n) \quad (9a)$$

in eqns.(8) and (9), respectively. Please refer to Appendix A.

Therefore, for $|m| > n$ (outside of polynomial coefficients window), we can obtain an unified recursive relations:

$$S_m + a_1 S_{m-1} + \cdots + a_n S_{m-n} = 0, \quad (|m| > n) \quad (10)$$

The above recursive relationships in eqns.(8) and (9) are named as Newton identities.

From these Newton identities, we can obtain a theorem as follows:

Theorem 1: Suppose that an n order polynomial described by eqn (1) is known, then a set of parameters (root moment) $\{S_m, m=1,2,\dots,n\}$ is solely determined recursively through eqn (8). Conversely, given the n root moments $\{S_m, m=1,2,\dots,n\}$, an n order polynomial like eqn (1) is solely determined recursively through eqn (8).

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

Corollary 1: Suppose that an n order polynomial described by eqn (1) is known, then a set of parameters (root moment) $\{S_m, m=-1,-2,\dots,-n\}$ is solely determined recursively through eqn (9). Conversely, given the n root moments $\{S_m, m=-1,-2,\dots,-n\}$, an n order polynomial like eqn (1) is solely determined recursively through eqn (9).

After discussing the constrained conditions implicit in polynomials, we present simply the constrained learning algorithm proposed by

Perantonis, et.al and Hormis, et.al [5,6] in the following subsection.

3.2 Constrained Learning Algorithm

For the first model of eqn.(4)¹, suppose that P training patterns are selected from the region $|x| < 1$, an error cost function is defined at the output of the network:

$$E(w) = \frac{1}{2P} \sum_{p=1}^P e_p^2(w) = \frac{1}{2P} \sum_{p=1}^P (o_p - y_p)^2 \quad (11)$$

where w is the set of all weights in the network model, $o_p = f(x_p)$ denotes the target (outer-supervised) signal to be found roots, $y_p = \prod_{i=1}^n (x_p - w_i)^2$ denotes the actual output of the network, $p = 1, 2, \dots, P$ is an index labeling the training patterns.

According to the above additional information (constrained conditions) available defined in eqn.(6) or eqn.(8) or eqn.(9), we can unifiedly write them as follows:

$$\Phi = 0 \quad (12)$$

where $\Phi = [\Phi_1, \Phi_2, \dots, \Phi_m]^T$ ($m \leq n$) (T denotes the transpose of a vector or matrix) is a vector composed of the constraint conditions of eqn. (6) or eqn.(8) or eqn.(9).

By considering the error cost function, which possibly contains many long narrow troughs, a constraint for updated synaptic weights is imposed in order to avoid missing the global minimum. Consequently, the sum of square of the individual weight changes takes a predetermined positive value $(dP)^2$:

$$\sum_{i=1}^n (dw_i)^2 = (dP)^2 \quad (13)$$

where dw_i denotes the change of synaptic weight w_i , dP 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 dP centered at the point defined by the current weight vector. If dP is small enough, the change to $E(w)$ and to Φ induced by changes in the weights can be approximated by the first differentials $dE(w)$

and $d\Phi$.

In order to derive the CLA based on the constraint conditions of eqns (8) or (9) and (13), assume that $d\Phi$ is equal to a predetermined vector quantity dQ , designed to bring Φ closer to its target (zero). The objective of learning process is to ensure that the maximum possible change in $|dE(w)|$ is achieved at each epoch. Usually, the maximization of $|dE(w)|$ can be carried out analytically by introducing suitable Lagrange multipliers. Thus, a vector $V = [n_1, n_2, \dots, n_m]^T$ of Lagrange multipliers is needed to take into account the constraints in eqn (12). Another Lagrange multiplier m is introduced for eqn (13).

By introducing the function e , de is expanded as follows [5,6]:

$$de = \sum_{i=1}^n J_i dw_i + (dQ^T - \sum_{i=1}^n dw_i F_i^T) V + m \left[(dP)^2 - \sum_{i=1}^n (dw_i)^2 \right] \quad (14)$$

$$\text{where } J_i = \frac{\partial E(w)}{\partial w_i} = \frac{1}{P} \sum_{p=1}^P e_p(w) \cdot \prod_{j \neq i} (x_p - w_j),$$

$$F_i = [F_i^{(1)}, F_i^{(2)}, \dots, F_i^{(m)}]^T, \quad F_i^{(j)} = \frac{\partial \Phi_j}{\partial w_i} \quad (i = 1, 2, \dots, n, j = 1, 2, \dots, m), \quad m \leq n$$

denotes the number of the constraint conditions in eqn (12).

Further, to maximize $|de|$ (in fact, minimize de) at each epoch, we demand that:

$$d^2 e = \sum_{i=1}^n (J_i - F_i^T V - 2m dw_i) d^2 w_i = 0 \quad (15)$$

$$d^3 e = -2m \sum_{i=1}^n (d^2 w_i)^2 < 0 \quad (16)$$

As a result, the coefficients of $d^2 w_i$ in eqn (14) should vanish, i.e.,

$$dw_i = \frac{J_i}{2m} - \frac{F_i^T V}{2m} \quad (17)$$

where the values of Lagrange multipliers m and V can be readily evaluated from eqns (13) and (17) and the condition $(dQ^T - \sum_{i=1}^n dw_i F_i^T) V = 0$ is embodied in eqn (13) with the following results:

¹ The fundamental rationale for deriving the CLA is the same for the second model of eqn.(5)

$$\mathbf{m} = -\frac{1}{2} \left[\frac{I_{JJ} - I_{JF}^T I_{FF}^{-1} I_{JF}}{(\mathbf{dP})^2 - \mathbf{dQ}^T I_{FF}^{-1} \mathbf{dQ}} \right]^{1/2} \quad (18)$$

$$V = -2\mathbf{m} I_{FF}^{-1} \mathbf{dQ} + I_{FF}^{-1} I_{JF} \quad (19)$$

where $I_{JJ} = \sum_{i=1}^n J_i^2$ is a scalar, I_{JF} is a vector whose components are defined by:

$$I_{JF}^{(j)} = \sum_{i=1}^n J_i F_i^{(j)}, \quad j=1,2,\dots,m \quad (20)$$

Specifically, I_{FF} is a matrix, whose elements are defined by $I_{FF}^{jk} = \sum_{i=1}^n F_i^{(j)} F_i^{(k)}$ ($j,k=1,2,\dots,m$).

Obviously, there are $(m+1)$ parameters $\mathbf{dP}, \mathbf{dQ}_j$ ($j=1,2,\dots,m$), which need to be set before the leaning process begins. Parameter \mathbf{dP} is often selected as a fixed value. However, the vector parameters \mathbf{dQ}_j ($j=1,2,\dots,m$) are generally selected as proportional to Φ_j , i.e., $\mathbf{dQ}_j = -k\Phi_j$ ($j=1,2,\dots,m$ and $k>0$), which ensures that the constraints Φ move towards zero at an exponential rate as training progresses [5,6].

From eqn (17), we note that k should satisfy $k \leq \mathbf{dP} (\Phi^T I_{FF}^{-1} \Phi)^{-1/2}$. In practice, the simplest choice

for k is $k = \mathbf{hdP} / \sqrt{\Phi^T I_{FF}^{-1} \Phi}$, where $0 < h < 1$ is another free parameter of the algorithm apart from \mathbf{dP} .

4 Experimental Results

In order to verify the effectiveness and efficiency of the above approaches, several experimental results are presented in this section. Here, we report the experimental results for one of these problems.

Example: $f(x) = x^4 + 2.9x^3 - 0.8x^2 - 6.3x - 3.6$.

For this 4-order polynomial, the four real roots are known as $w_1 = -1.0$, $w_2 = -1.0$, $w_3 = 1.5$ and $w_4 = -2.4$, respectively. Now we use neural network approach to solve this problem. The corresponding real roots can be found by the roots moment method. The constrained learning algorithm introduced in Section 3 is used to train the corresponding neural networks until the network converges (the termination error is assumed to be 1.0×10^{-9}). As a result, the learning (error and weight) curves are shown in Figs 2-6. The four estimated real roots are $w_1 = -1.00003$, $w_2 = -0.99987$, $w_3 = 1.49986$ and $w_4 = -2.40005$, respectively. Experimental results show that our approach can perform much faster training and achieve good accuracy.

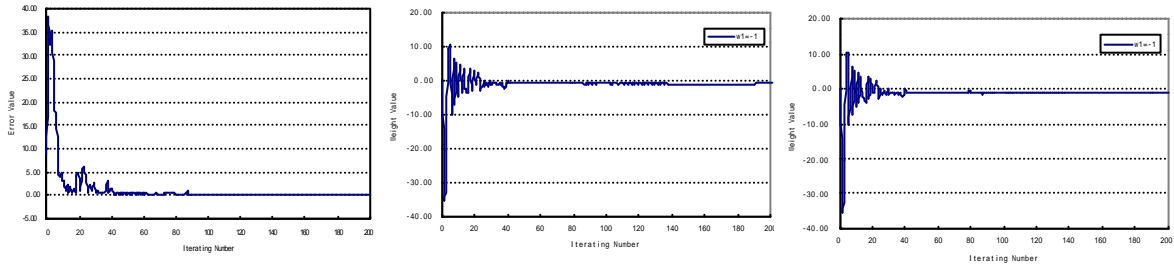


Fig.2 The learning error curve Fig.3 The learning weight (root) curve ($w_1 = -1$). Fig.4 The learning weight (root) curve ($w_4 = -2.4$).

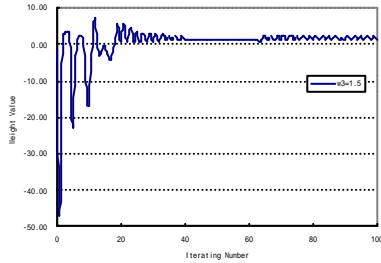


Fig. 5 The learning weight (root) curve ($w_3 = 1.5$).

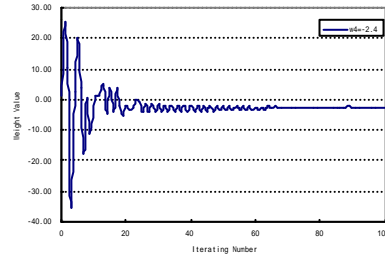


Fig.6 The learning weight (root) curve ($w_4 = -2.4$).

5 Conclusions

The paper proposed using feedforward neural networks trained by the roots moment based

constrained learning algorithm to find the roots of polynomials. It is found that this roots moment approach can rapidly obtain the solutions of the problem. Specifically, for high order polynomials with plenty of computations, the roots moment approach will exhibit its potential. The experimental results support our claims.

Appendix A

The derivation of eqn. (9) is given as follows.

We write eqn (4) as follows:

$$f(z) = \prod_{i=1}^n (z - w_i) \quad (4)$$

hence, we can obtain the derivative of $f(z)$ w.r.t. z :

$$f'(z) = \sum_{i=1}^n \frac{f(z)}{z - w_i} \quad (A.1)$$

Since w_i^m ($m < 0$) need to be computed in considering

S_m ($m < 0$), so the term $\frac{1}{z - w_i}$ in eqn. (A.1) can be

expanded in the interval of $|z| \leq w_i$ according to the following form:

$$\begin{aligned} \frac{1}{z - I_i} &= -\frac{1}{I_i} \cdot \frac{1}{1 - zI_i^{-1}} = -I_i^{-1}(1 + zI_i^{-1} + \dots) \\ &= -(I_i^{-1} + zI_i^{-2} + z^2I_i^{-3} + z^3I_i^{-4} + \dots) \end{aligned} \quad (A.2)$$

Substituting eqns. (1) and (A.2) into eqn.(A.1):

$$\begin{aligned} f'(z) &= -\sum_{i=1}^n f(z) \cdot (I_i^{-1} + zI_i^{-2} + z^2I_i^{-3} + \dots) \\ &= -f(z) \cdot (S_{-1} + zS_{-2} + z^2S_{-3} + \dots) \\ &= -(z^n + a_1z^{n-1} + \dots + a_{n-1}z + a_n) \cdot (S_{-1} + zS_{-2} + \dots) \\ &= -a_nS_{-1} - (a_{n-1}S_{-1} + a_nS_{-2})z - \dots \\ &\quad - (S_{-1} + a_1S_{-2} + a_2S_{-3} + \dots + a_nS_{-n-1})z^n - \dots \end{aligned} \quad (A.3)$$

According to eqn.(1), we can derive another expression of $f'(z)$ as follows:

$$\begin{aligned} f'(z) &= nz^{n-1} + (n-1)a_1z^{n-2} + (n-2)a_2z^{n-3} + \dots \\ &\quad + (n-m)a_mz^{n-m-1} + \dots + 2a_{n-2}z + a_{n-1} \end{aligned} \quad (A.4)$$

By comparing eqn.(A.3) with each term of eqn.(A.4), we have an unified expression:

$$\begin{aligned} a_nS_m + a_{n-1}S_{m+1} + \dots + a_{n+m+1}S_{-1} + ma_{n+m} &= 0, \\ (-1 \geq m \geq -n, a_0 = 1) \end{aligned} \quad (A.5)$$

For those terms of z^i ($i \geq n$) in eqn. (A.3), which correspond to 0's in eqn. (A.4), we have:

$$a_nS_m + a_{n-1}S_{m+1} + \dots + S_{m+n} = 0, \quad (m < -n) \quad (A.6)$$

Therefore, combining eqn.(A.5) and eqn.(A.6), we obtain:

$$\begin{cases} a_nS_{-1} + a_{n-1} = 0 \\ a_nS_{-2} + a_{n-1}S_{-1} + 2a_{n-2} = 0 \\ \vdots \\ a_nS_m + a_{n-1}S_{m+1} + \dots + ma_{n+m} = 0, \quad (m \geq -n, a_0 = 1) \\ a_nS_m + a_{n-1}S_{m+1} + \dots + S_{m+n} = 0, \quad (m < -n) \end{cases} \quad (A.7)$$

which is just eqn.(9).

If we replace m of eqn.(8a) with $m+n$, eqn.(8a) will become into eqn.(9a). In other words, eqn.(8a) and eqn.(9a) are completely equivalent.

References

- [1] D.S.Huang, Zheru Chi, "Neural Networks with Problem Decomposition for Finding Real Roots of Polynomials", IJCNN2001 (accept).
- [2] M. Lang and B.-C. Frenzel, "Polynomial root finding," *IEEE Signal Processing Letters*, Vol.1, No.10, pp.141-143,1994.
- [3] T. Stathaki, Root moments: "a digital signal-processing perspective," *IEE Proc. Vis. Image Signal Processing.*, 145, 293-302,1998.
- [4] R.Hormis, G. Antonion and S. Mentzelopoulou, "Separation of two-dimensional polynomials via a $\Sigma - \Pi$ neural net," in *Proceedings of International Conference on Modelling and Simulation*, Pittsburg, PA, USA, pp.304-306, 1995.
- [5] D.A. Karras and S.J. Perantonis, "An efficient constrained training algorithm for feedforward networks," *IEEE Trans. Neural Networks*, Vol.6, pp.1420-1434, 1995.
- [6] S.J. Perantonis and D.A. Karras, "An efficient constrained learning algorithm with momentum acceleration," *Neural Networks*, Vol.8, pp.237-249, 1995.
- [7] Dezhi Fang, et.al, *Handbook of Mathematics*. Publishing House of High Education (in Chinese), Beijing, China, pp.87-116, 1979.