FINDING THE MAXIMUM MODULUS ROOTS OF POLYNOMIALS BASED ON CONSTRAINED NEURAL NETWORKS[1]

De-Shuang Huang[1,2] Horace H.S.Ip[2]

1 Institute of Intelligent Machines, Chinese Academy of Sciences,
P.O.Box 1130, Hefei, Anhui 230031, China
2 AIM*tech* Centre, Department of Computer Science, City University of Hong Kong,
83 Tat Chee Avenue, Kowloon Tong, Hong Kong,
emails: huangdeshuang@yahoo.com, cship@cityu.edu.hk

ABSTRACT

This paper focuses on how to find the Maximum Modulus Root (MMR) (real or complex) of an arbitrary polynomial. Efficient solution to this problem is important for many fields including neural computation and digital signal processing etc.. In this paper we present neural networks technique for solving this problem. Our Neural Root Finder (NRF) is designed based on partitioning Feedforward Neural Networks (FNN) trained with a Constrained Learning Algorithm (CLA) by imposing the *a priori* information about the root moment from polynomial into the error cost function. Experimental results show that this neural root-finding method is able to find the maximum modulus roots of polynomials rapidly and efficiently.

1. INTORDUCTION

Since finding the roots of polynomials is important for many areas of signal processing such as spectral factorization, phase unwrapping, forming a cascade of lower order systems, and so on, some works have been reported on finding the roots of polynomials. Literature [1] gives a comprehensive review of polynomial root-finding methods that include the eigenvalue-based method, the Jenkins-Traub approach, etc. Most of them, however, adopted the conventional iterating or recursive methods. Moreover, almost all methods had difficulty in achieving both accuracy and processing speed [1,2].

In 2000, L.Hoteit proposed using the FFT-based differential cepstrum estimation for computing the roots of a moderate to high order complex polynomial [2]. The root-finding method consists of an iterative estimation of the roots, followed by an estimation of the associated roots through an FFT-based on factorization. By means of this FFT-based differential cepstrum estimation ones can find the maximum root modulus of a polynomial. Nevertheless, the method for finding the polynomial maximum root modulus needs to estimate the slope of the curve about the logarithm of the modulus of the *mth* root moment of a polynomial, $\log(|S_m|)$, against the order of the root moment m. Obviously, the corresponding accuracy is limited, and the computational complexity is also increased. Specifically, this method fails to find the phase information of the MMR of polynomial, which is sometimes vital to signal processing.

Inspired by the CLA by Perantonis, et.al and Hormis, et.al for training FNN for factorizing 2-D Polynomials [3], we designed a CLA [4] to train an FNN root-finder by imposing the constrained relations between the roots and the coefficients of a polynomial into the batch-style error cost function. To further save computation load, we propose a new structural neural network model, referred to as the Partitioning Recursive Neural Root Finder of Polynomial (PRNRFP), for recursively obtaining a few roots at a given time [4]. Moreover, we extended the case of finding real roots of polynomial to a general case of finding complex roots of polynomial [5].

In this paper we will discuss using the PRNRFP model imposed by the *a priori* information of Root Moments (RM) from polynomial to find the maximum modulus roots of arbitrary polynomials. Simulation results will be presented to support our approach.

2. THE PARTITIONING RECURSIVE NEURAL ROOT FINDER OF POLYNOMIAL

An n order arbitrary polynomial $f(x)$ can be denoted as $f(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$, where $n \geq 2, a_0 \neq 0$. Without loss of generality, the coefficient a_0 of x^n is usually set as 1. Suppose that $f(x)$ is factorized into the following form:

$$f(x) = (x - w_1)(x - w_2)\cdots(x - w_i) f_b(x) \qquad (1)$$

where $w_1, w_2, \cdots, w_i$ are the i synaptic weights, i.e., the i roots to be found, the remaining polynomial $f_b(x)$ with an order of $(n - i)$ can be expressed as $f_b(x) = x^{n-i} + b_1 x^{n-i-1} + \cdots + b_{n-i}$, where $\{b_1, b_2, \cdots, b_{n-i}\}$ are the coefficients of the remaining polynomial $f_b(x)$. The structure sketch for this PRNRFP [4] is shown in Fig.1. Obviously, when $i = 1$, we can get one root at a time. In the following, we will discuss how to use the PRNRFP with $i = 1$

[1] This work was supported by NSFC of China.

based on the RM' s to get the maximum modulus root.

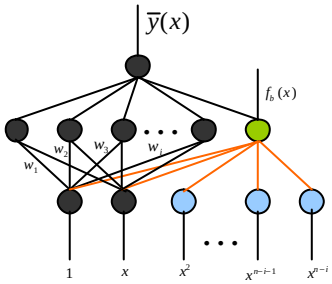


Fig.1 Partitioning recursive neural root finder architecture for finding i roots at a time of polynomials

3. THE ROOTS MOMENT AND CONSTRAINED LEARNING ALGORITHM

3.1. The Root Moment

The concept of the RM' s of polynomial was first formulated by Isaac Newton [6], which is defined as follows:

Definition 1: For an n order polynomial $f(x)$, assume that the corresponding n roots are respectively $w_1, w_2, \cdots,$ and w_n, then, the m ($m \in Z$) order RM of polynomials is defined as

$$S_m = w_1^m + w_2^m + \cdots + w_n^m = \sum_{i=1}^{n} w_i^m \tag{2}$$

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

$$\begin{cases} S_1 + a_1 = 0 \\ S_2 + a_1 S_1 + 2a_2 = 0 \\ \vdots \\ S_m + a_1 S_{m-1} + \cdots + a_n S_{m-n} = 0, \quad (m > n) \end{cases} \tag{3}$$

In fact, we can prove eqn.(3) is equivalent to:

$$S_m + a_1 S_{m-1} + \cdots + a_n S_{m-n} = 0, \qquad (m > n) \tag{4}$$

The above recursive relationships in eqns.(3) and (4) are named as Newton identities. From these Newton identities, we can obtain a theorem as follows:

Theorem 1: Suppose that an n order polynomial $f(x)$ is known, then, a set of parameters (the RM' s) $\{S_m, m = 1,2,\cdots,n)$ is uniquely determined recursively through eqn (4). Conversely, given the n RM' s $\{S_m, m = 1,2,\cdots,n)$, an n order polynomial $f(x)$ is uniquely determined recursively through eqn (4).

After discussing the constrained conditions implicit in polynomials, we present simply the complex CLA in the following subsection.

3.2. Complex Constrained Learning Algorithm

Suppose that P training patterns are selected from the region $|x| < 1$, an Error Cost Function (ECF) is defined at the output of the network:

$$E(w) = \frac{1}{2P} \sum_{p=1}^{P} |e_p(w)|^2 = \frac{1}{2P} \sum_{p=1}^{P} (o_p - y_p)(o_p - y_p)^* \tag{5}$$

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

According to the above additional information (constrained conditions) available from eqn.(3) or (4), we can unifiedly write them as follows:

$$\Phi = 0 \tag{6}$$

where $\Phi = [\Phi_1, \Phi_2, \cdots, \Phi_m]^T$ $(m \le n)$ (T denotes the transpose of a vector or matrix) is a vector composed of the constraint conditions of eqn. (3) or (4).

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

$$\sum_{i=1}^{n} |dw_i|^2 = (dP)^2 \tag{7}$$

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 the light of the ECF of eqn.(5) and the two constrained relations of eqns.(6) and (7), by imposing the two additional constraint relations into the ECF and by introducing suitable Lagrange vector and scale multipliers, V and m. Considering a small change from dw_i, a new ECF including two additional constraints is defined as follows:

$$de = dE(w) + (dQ^H - d\Phi^H)V + m\left[(dP)^2 - \sum_{i=1}^{n} |dw_i|^2\right] \tag{8}$$

By expanding the terms on the right hand side of eqn (8), we easily obtain:

$$de = \sum_{i=1}^{n} J_i dw_i + (dQ^T - \sum_{i=1}^{n} dw_i F_i^T)V + m\left[(dP)^2 - \sum_{i=1}^{n} (dw_i)^2\right] \tag{9}$$

where $J_i = \partial E(w)/\partial w_i = 1/P \sum_{p=1}^{P} e_p(w)/|x_p - w_i|$,

$F_i = [F_i^{(1)}, F_i^{(2)}, \cdots, F_i^{(m)}]^T$, $F_i^{(j)} = \dfrac{\partial \Phi_j}{\partial w_i}$

($i = 1,2,\cdots,n$, $j = 1,2,\cdots,m$). Similar to the derivation of [4], we can derive the following relation:

$$dw_i = \frac{J_i}{2m} - \frac{F_i^H V}{2m} \tag{10}$$

where

$$m = -\frac{1}{2}\left[\frac{I_{JJ} - I_{JF}^H I_{FF}^{-1} I_{JF}}{(dP)^2 - dQ^H I_{FF}^{-1} dQ}\right]^{1/2} \tag{11}$$

$$V = -2mI_{FF}^{-1} dQ + I_{FF}^{-1} I_{JF} \tag{12}$$

where $I_{JJ} = \sum_{i=1}^{n} |J_i|^2$ is a scalar, I_{JF} is a vector whose components are defined by $I_{JF}^{(j)} = \sum_{i=1}^{n} J_i F_i^{(j)}$,($j = 1,2,\cdots,m$). Specifically, I_{FF} is a matrix, whose elements are defined by $I_{FF}^{jk} = \sum_{i=1}^{n} F_i^{(j)} F_i^{(k)}$ ($j,k = 1,2,\cdots,m$).

Generally, the parameter dP is adaptively selected as follows:

$$dP(t) = dP_0(1 - e^{-\frac{q_p}{t}}) \tag{13}$$

where dP_0 is the initial value for dP , which is usually chosen with a larger value; t is the time index; q_p is the scale coefficient of time t , which is usually set as $q_p > 1$.

However, the vector parameters $dQ_j (j = 1,2,\cdots,m)$ are generally selected as proportional to Φ_j , i.e., $dQ_j = -k\Phi_j$ ($j = 1,2,\cdots,m$, $k > 0$), which ensures that the constraints Φ move towards zero at an exponential rate as the training progresses [4,5].

4. EXPERIMENTAL RESULTS

To use the above CLA to train the PRNRFP with $i = 1$ for finding the maximum modulus root of a polynomial, and verify the effectiveness and efficiency of our approach, two experimental results are presented in this section.

In the following two examples, assume that the controlling parameter with the CLA is all chosen as $\{dP_0 = 1.0, e = 0.7, q_p = 5.0\}$, and 30 repeating experiments are respectively conducted by choosing different initial random weight values from the uniform distribution in $[-1,1]$ to observe the statistical experimental results. For each experiment, we first adopt the PRNRFP to find out all roots by letting the termination error accuracy being $e_r = 0.01, 0.001$, respectively. At the same time, the MMR's are selected out

Example 1: $f_1(x) = x^4 + 2x^3 - 2.3x^2 + 4.7x - 5.6$. For this 4-order polynomial, we adopt the PRNRFP with $i = 1$ to obtain the roots distributions in the complex planes for two accuracy cases, as shown in Fig.2. At the same time, the 10 selected MMR's and the corresponding Average Iterating Number (AIN) are shown in Table 1.

Example 2: $f_2(x) = x^6 + 2.1x^5 + -1.5ix^3 + 4.2x^2 + 3$. Likewise, for this 6-order polynomial, we also adopt the PRNRFP with $i = 1$ to obtain the roots distributions in the complex planes for two accuracy cases, as shown in Fig.3. Table 2 shows the 10 selected MMR's out and the corresponding AIN's.

The experimental results and performance statistics for the above two examples shows that the PRNRFP with $i = 1$ at each repeating trial can indeed select out the maximum modulus roots even when the initial weights at each trial are randomly chosen from [-1,1].

5. CONCLUSIONS

This paper proposed applying the partitioning recursive neural root finder of polynomial (PRNRFP) trained by the root moments based constrained learning algorithm for finding the maximum modulus root of an arbitrary polynomial. The statistical experimental results show that this recursive NRF approach based on the root moment constraints can indeed rapidly obtain the maximum modulus roots. Further study will include how to find the minimum modulus roots and the ordered roots of arbitrary polynomial.

.

6. REFERENCES

[1] A.W.Press, S.Teukolsky, W.Vetterling and B.Flannery. *Nemerical Recipes in C, the Art of Scientific Computing.* Cambridge University Press, 1988.

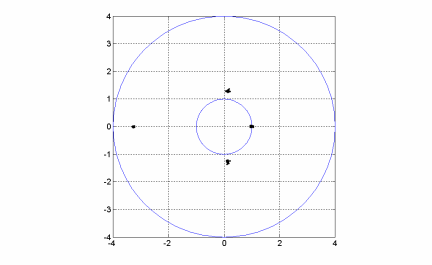
[2] L.Hoteit, "FFT-based fast polynomial rooting," *2000 IEEE International Conference on Speech, and Signal Processing (ICASSP '00), Proceedings*, Vol.6, pp.3315 –3318, 2000.

[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, Zheru Chi, "Neural networks with problem decomposition for finding real roots of polynomials," *2001 Int. Joint Conf. On Neural Networks (IJCNN2001)*, Washington, DC, Vol. Addendum, 25-30, July 15-19, 2001.

[5] D.S.Huang, Zheru Chi, "Finding complex roots of polynomials by feedforward neural networks," *2001 Int. Joint Conf. On Neural Networks (IJCNN2001)*, Washington, DC, Vol. Addendum, 13-18, July 15-19, 2001.

[6] T. Stathaki, "Root moments: a digital signal-processing perspective," *IEE Proc. Vis. Image Signal Processing*, 145, 293-302, 1998.



(a) $e_r = 10^{-2}$ (b) $e_r = 10^{-3}$

Fig. 2 The estimated root distributions of polynomial $f_1(x)$ for two cases of $e_r = 10^{-2}$ and $e_r = 10^{-3}$

(a) $e_r = 10^{-2}$ (b) $e_r = 10^{-3}$

Fig. 3 The estimated root distributions of polynomial $f_2(x)$ for two cases of $e_r = 10^{-2}$ and $e_r = 10^{-3}$

Table1 The selected 10 maximum modulus roots distributions for $f_1(x)$

Indices	The Maximum Modulus Roots	AIN
$e_r = 10^{-2}$	(-3.2876, 1.2614E-04) (-3.2938, -1.8535E-03) (-3.2724, -1.5103E-03) (-3.2920, 4.2354E-05) (-3.2923, 3.5135E-04) (-3.2871, -5.4411E-05) (-3.2594, -2.5273E-03) (-3.2935, -7.9741E-04) (-3.3000, -4.1932E-03) (-3.2995, -2.7129E-03)	26
$e_r = 10^{-3}$	(-3.2905, 1.2664E-04) (-3.2914, -5.3297E-04) (-3.2896, -7.7427E-05) (-3.2905, 2.5303E-06) (-3.2905, 5.8580E-05) (-3.2891, -1.0246E-04) (-3.2907, 1.6246E-05) (-3.2909, -1.1754E-04) (-3.2913, -4.3486E-04) (-3.2905, -4.1454E-05)	52

Table2 The selected 10 maximum modulus roots distributions for $f_2(x)$

Indices	The Maximum Modulus Roots	AIN
$e_r = 10^{-2}$	(-2.0458, 0.6697) (-2.0381, 0.6632) (-2.0450, 0.6694) (-2.0539, 0.6763) (-2.0510, 0.6591) (-2.0363, 0.6742) (-2.0480, 0.6700) (-2.0479, 0.6708) (-2.0482, 0.6759) (-2.0450, 0.6695)	114
$e_r = 10^{-3}$	(-2.0452, 0.6693) (-2.0450, 0.6725) (-2.0452, 0.6694) (-2.0482, 0.6717) (-2.0470, 0.6664) (-2.0422, 0.6710) (-2.0431, 0.6689) (-2.0479, 0.6708) (-2.0459, 0.6706) (-2.0450, 0.6695)	150