Successive Adaptive Linear Neural Modeling for Equidistant Real Roots Finding

Joseph R. González
School of Mathematics and Computer Science
University of Yachay Tech
San Miguel de Urcuquí, Ecuador
joseph.gonzalez@yachaytech.edu.ec

Fernando P. Zhapa
School of Mathematics and Computer Science
University of Yachay Tech
San Miguel de Urcuquí, Ecuador
fernando.zhapa@yachaytech.edu.ec

Oscar V. Guarnizo
School of Mathematics and Computer Science
University of Yachay Tech
San Miguel de Urcuquí, Ecuador
oscar.guarnizo@yachaytech.edu.ec

Francisco Ortega-Zamorano
Department of Computer Languages and Computer Science
University of Málaga
Málaga, Spain
fortega@lcc.uma.es

***Abstract*—The main objective of this work has been to implement a model to find equidistant real roots using a Successive Adaptive Linear Neural Modeling which uses two approaches: a Self Organized Map (SOM) and an Adaptative Linear Neuron (Adaline). A SOM model has been used with a new neighborhood function Λ, and a physical distance β with which the task is divided in sub-processes reducing the complexity of the task because the SOM model can delimited the areas where a single root exist. Then, through a successive approach, it is applied an Feed-forward neural model with a learning process base on Adaline neuron with pocket in each pair of regions for finding the real root values with a reduced precision. Finally, several experiments were done consider CPU time, relative error, distance between the roots and polynomial degrees. The results show that the time complexity grows in a linear or logarithmic way. Also, the error does not increase in a higher rate than the degree of polynomial or the root distance.**

***Index Terms*—Polynomial roots, Supervised learning, SOM.**

I. Introduction

The polynomial roots finding is one of the classical problems in mathematics due to its large use in many mathematical processes such as eigenvalues finding, differential equations resolution, etc. This problem was known in the third millennium B.C. by the Sumerians, and promoted the research of some mathematical concepts such as irrational and complex numbers, algebraic groups, fields, ideals and numerical computing [1].

There are many applications in which it is necessary to solve polynomials, such as the signal processing, communication (coding theory and decoding theory), etc [2]. The methods that have usually been used to resolve these problems are linear algebra, linear programming, Fast Fourier transform (FFT) [1]. In most of the cases, these problems imply to solve easily polynomials with smaller degrees, but when the degree increases the problem is more complex. In fact, studies in polynomial roots finding have reached formulas to get roots in analytical way until fourth-degree polynomials. After that, there were several non-successful attempts until Abel in 1824 demonstrated the unsolvability of polynomials of fifth-degree and higher in terms of radicals [3]. Then, Galois in 1830 went further and gave conditions for which some particular types of polynomials can have a formula [4].

Polynomials greater than fifth grade are not solvable with an analytical procedure, for this reason numerical methods are so important because they are useful to find roots of polynomials. There are a lot of numerical methods, but the most popular are: bisection, secant, fixed-point and Newton-Raphson and iterative methods that use matrix algebra such as power method, QR algorithm, Singular Value Decomposition (SVD) [5].

The problem of numerical methods is that, in the majority of cases, it is needed to know a initial value (a first perception of the solution) and it can be only got one root at a time[6]. For this reason, it has motivated the research of more efficient algorithms which can resolve this problem with less resources and faster. In this context, alternatives have been sought to solve polynomials through artificial neural networks.

Artificial Neural Networks (ANN) are mathematical models inspired by the functioning of the brain of living beings that have shown to have interesting application in several practical domains, in particular to find roots of polynomials. In this field of work, it has found some approaches such as the use of Artificial Feed Forward Neural Networks (FNN) [7] [8], Multilayer Perceptron Network (MLPN) using constrained learning algorithm (CLA) [2], etc. On one hand, in [8], it was implemented an FNN with two hidden layers with an architecture Z-8-7-Y where Z represents the number of

978-1-5386-6657-9/18/$31.00

coefficients and Y represents the number of classes. In the hidden layer, it is found the number of real roots, and then the approximated roots are obtained in the output layer. A fundamental result of this method is that all the real roots can be obtained at the same time in parallel. It is an advantage in front of other current approaches which perform a sequential procedure to find polynomial roots, in other words, the current procedures can not continue until the previous root has been found. On the other hand, in [2] and [7], the roots were found using a Constrained Learning Algorithm (CLA) that uses *a priori* information of the polynomial. In [7] *a priori* information are only the root moments and in [2] the *a priori* information is the relation between the root moments and coefficients of the polynomial. The CLA approach makes possible to find (real or complex) roots of arbitrary (real or complex) polynomials.

In this work, it is proposed to find polynomial roots using a Self Organized Map (SOM) to get regions in which roots exist. When the regions are gotten, it is introduced the patterns into a Successive Adaptive Linear Neuron (Sadaline) with pocket in which, for every pair of regions, a neuron identifies the root. With this proposal, root finding problem is reduced to a clustering problem followed by a partition of regions. It is important to mention that the aim of this paper is to proposed other way to find real root, but not to compare the efficiency of this approach with others already existing.

II. Methodology

A. Model Proposal

In this paper, it is proposed the use of two ANN's to find the real roots of a real polynomial. The first ANN is a SOM that will receive the patterns and classify them into regions of possible roots. After that it is used a Sadaline with pocket to find the roots in those regions.

The topology of the SOM is simply linear (Fig. 1) because the patters are distributed around x axis and there are $n+1$ neurons because there are n points, in this case n roots dividing a whole will produce $n+1$ regions.

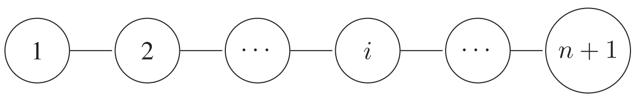

Fig. 1: Topology of SOM

In this SOM, it is implemented a neighborhood function which updates in each step five neurons, the winning neuron and the closest neurons of both sides two on the left and two on the right. In order to perform this training, the neurons are multiplied by values 1, 0.5, 0.1. In specific the neighborhood function is

$$\Lambda(d_r, d_i) = \begin{cases} 1 & if \quad |d_r - d_i| = 0 \\ 0.5 & if \quad |d_r - d_i| = 1 \\ 0.1 & if \quad |d_r - d_i| = 2 \\ 0 & otherwise \end{cases} \tag{1}$$

where d_r is the coordinate of the winning neuron over the x axis, d_i is the coordinate x of i-neuron and $|d_r - d_i|$ is the topological distance between to neurons.Then, any neuron is updated with the following training function

$$w_i(k+1) = w_i(k) + \eta(k)\Lambda(d_r, d_i)(x(k) - w_i(k) \tag{2}$$

The other network used is a simple Adaptive Linear Neuron (Adaline). It is launched n times in order to find the n respective roots. Moreover, for improving the accuracy of the result neurons, the network is used a pocket to stored the weight of neuron with the best accuracy.

It must be said that this modeling only works for real equidistant roots. That is, given a polynomial

$$p_n(t) = t^n + a_{n-1}t^{n-1} + \cdots + a_2t^2 + a_1t^1 + a_0 \tag{3}$$

it can be written in the form

$$p(t) = \prod_{i=1}^{n}(t - \lambda_i) \tag{4}$$

where the root distance $r = |\lambda_i - \lambda_{i+1}| = k$

Since a whole is being divided into regions, another constraint for this modeling is that the roots must not have multiplicity greater than 1.

B. Explained Algorithm

Given a polynomial

$$p_n(t) = t^n + a_{n-1}t^{n-1} + \cdots + a_2t^2 + a_1t^1 + a_0 \in K[t];$$

K denotes the coefficient field ($\mathbb{R}$) of the polynomials ring and n denotes the degree of the polynomial.

The algorithm stars generating patterns around the x axis of the plane, then the algorithm evaluates each pattern on the polynomial function and decide whether of them is above or below the function, putting the points above the function in the class '1' and for the points below the function the class '-1'.(see the Fig. 2).

In this picture (see the Fig. 2). the tags '1' and '-1' are just identifiers of classes. Nevertheless, when we use a SOM to find the regions of classifications these tags become a new dimension, it is useful to separate physically the two classes '1' and '-1' see the Fig. 3.

This distance can be multiplied by a factor β in order to change the performance of the SOM. Thanks to this distance the SOM can divide the set of points in different regions of classification. At the end the algorithm has divided $n+1$ regions (Fig. 4).

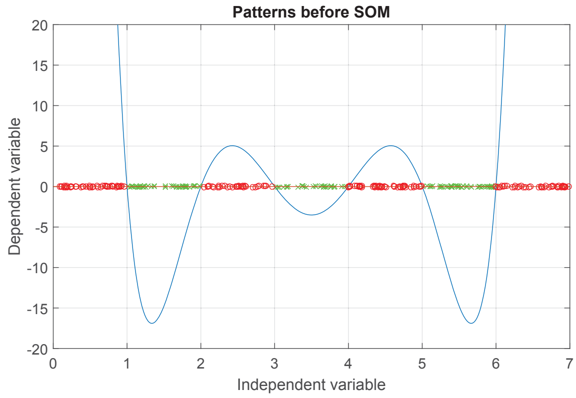

Fig. 2: Patterns before SOM

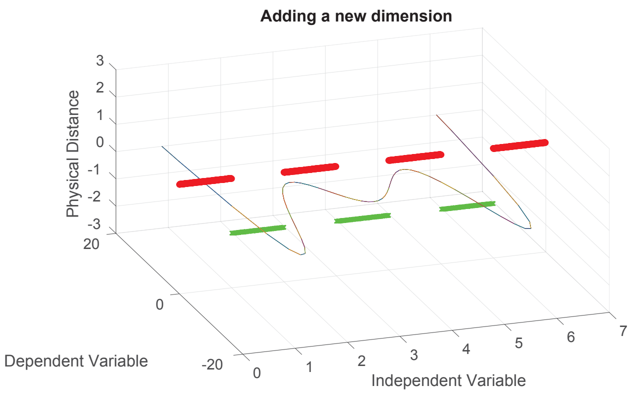

Fig. 3: Adding a new dimension

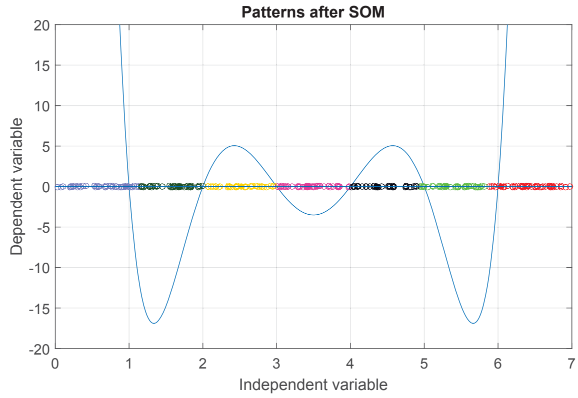

Fig. 4: Patterns after SOM

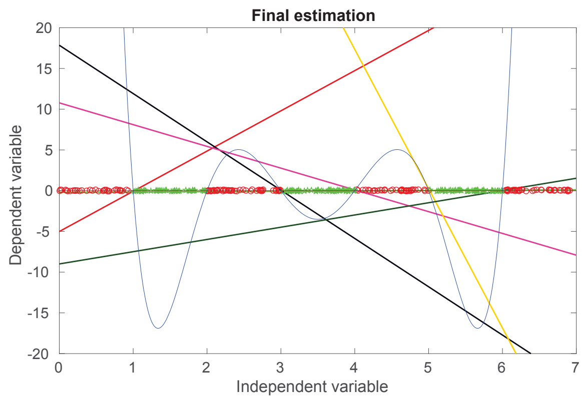

Fig. 5: Final Estimation

These regions are used in another ANN, the Sadaline, in which we enter the patters of the regions i and $i+1$ with a neuron, so the neuron divides the plane according the two classes ('1' and '-1') finding a root $\lambda_i \in \mathbb{R}$. At the end the the each neuron will place on each root as seen in Fig. 5

III. Results

For the analysis we have taken into consideration four important factors: CPU time, relative error, distance between the roots and polynomial degree.

The parameters for the SOM were:

- $\eta_0 = 0.7$
- $\beta = 0.25$
- Iterations = 400

The parameters for the Sadaline where:

- $\eta_0 = 0.01$
- Epochs = 100

Finally, the number of patterns used in all the tests were 300 randomly chosen. We have done test on several polynomials from different degrees and polynomials with the same degree but different root distance.

For Fig. 6 and Fig. 7 we chose polynomials from degree 1 to 10 with root distance $d = 1$. Every sample was tested 10 times.

Another aspect analyzed was the relation between relative error and distance between roots (Fig. 8). For this analysis it was chosen ten polynomials with 6 roots each one and distance from 1 to 10. Mathematically, the polynomials were:

$$p_i(t) = \prod_{j=1}^{6} (x - \lambda_{i,j}), i = 1, 2, ..., 10$$

and

$$r_i = |\lambda_{i,j} - \lambda_{i,j+1}| = i, j = 1, 2, ..., 5$$

IV. Discussion

In Fig. 6 it is possible to see that the complexity in time of the algorithm respect to the degree of the polynomial seems to be linear or logarithmic. The reason for that behavior is that the SOM is the determining factor for the complexity. Every launched Adaline has the same complexity in time, so at the end the complexity of all the Adalines will be the sum of the complexity of each Adaline. This makes our algorithm predictable and linear in that part. However, going back to the SOM, the complexity of the algorithm is decided there.

In Fig. 7 we can appreciate that the relative error tends to grow very slow when polynomial degree increases. In fact, we can see that for degree 5 the error is less than for degree 4.

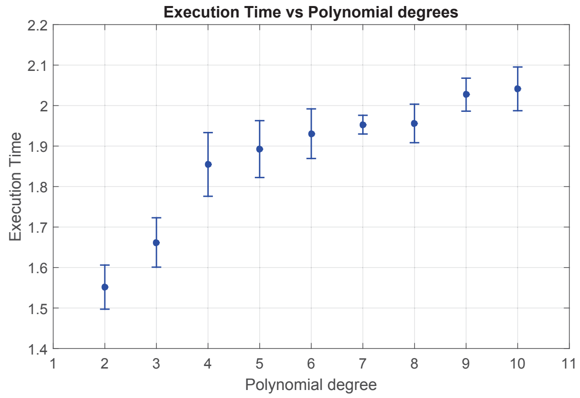

Fig. 6: Execution time vs. Degree of polynomial

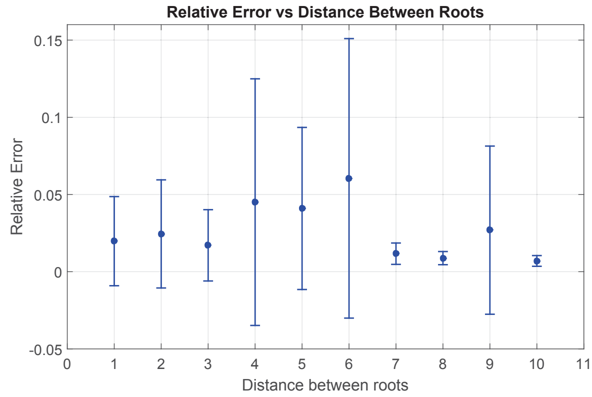

Fig. 8: Relative Error vs. Distance between roots

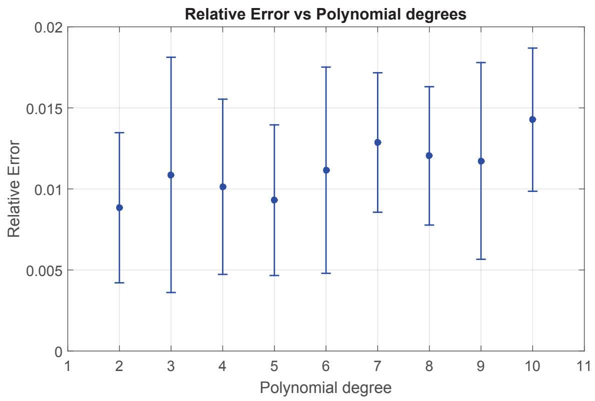

Fig. 7: Relative Error vs. Degree of polynomial

The same case can be seen with degrees 8 and 7. The reason also here is that the accuracy of the algorithm is decided in the SOM stage. When the SOM divides the patterns in *good* regions, the results in the Adaline's stage will be more accurate because every Adaline will be in a very good region to work. On the other hand, when the SOM does not divide correctly the regions, in the Adaline part there will be mistakes. Commonly, the mistakes that happen in the SOM part is that one or two regions are not well divided, therefore, in the Adaline part one or two roots will be mistaken respectively. That means that if the SOM fails it does not affect the whole result, only those particular roots associated with the wrong-divided regions.

When roots distance is larger, the patterns in the regions get scattered because they have more *space*. In those terms, it could be expected to increase the error when the root distance increases and the number of initial patterns does not change. However, in Fig. 8 that hypothesis is refused, because the error does not grow with the distance of polynomials. Probably the result is given by the randomness of the position of the patterns. Which is curious is that the standard deviation in distances 7, 8 and 10 are the smallest in the graph, whereas for degree 5 and 6 the standard deviation is the biggest. Despite the error does not grow as expected, the magnitude of those errors with respect to is bigger than in Fig. 7.

V. Conclusions

The obtained results of the model proposed in this paper show that ANN's can be used to find roots of polynomials. Other types of ANN have been proposed before, but all of them try to interpolate the polynomial, or make a regression. The model proposed here gives another perspective to the problem transforming it into a clustering problem followed by a simple separation of regions.The advantages of this model is that time complexity grows in a linear or logarithmic way. Also, the error does not increase in a higher rate than the degree of polynomial or the root distance. It is important to specified that the previous results were obtained using a sequential approach, but in order to obtain better results this algorithm can be implemented in a parallel way. It is possible because after finding the regions of the roots, it can be inserted and computed the neurons in each pair of regions at the same time. Despite of that, the model is very limited because it works only for polynomials with the same distance between its roots. However, it is expected that reviewing the SOM network, and changing its neighborhood to an appropriate one, could give the possibility to work with polynomials with different length between its roots.

References

[1] V. Y. Pan, "Solving a polynomial equation: some history and recent progress," *SIAM review*, vol. 39, no. 2, pp. 187–220, 1997.

[2] D.-S. Huang, "A constructive approach for finding arbitrary roots of polynomials by neural networks," *IEEE Transactions on Neural Networks*, vol. 15, no. 2, pp. 477–491, 2004.

[3] M. Rosen, "Niels hendrik abel and equations of the fifth degree," *The American Mathematical Monthly*, vol. 102, no. 6, pp. 495–505, jun 1995.

[4] J.-P. Tignol, *Galois' Theory Of Algebraic Equations*. World Scientific Publishing Company, 2001.

[5] R. L. Burden and J. D. Faires, *Numerical Analysis*, 9th ed. Brooks Cole, 2011.

[6] V. M. Foresi, *Curso de métodos numéricos*. Euskal Herriko Unibertsitatea, Argitalpen Zerbitzua, 1998.

[7] D.-S. Huang, H. H. Ip, and Z. Chi, "A neural root finder of polynomials based on root moments," *Neural Computation*, vol. 16, no. 8, pp. 1721–1762, 2004.

[8] M. Das and P. Seal, "Polynomial real roots finding using feed forward neural network: A simple approach," in *Computing and Communication Systems (NCCCS), 2012 National Conference on*. IEEE, 2012, pp. 1–4.

TABLE I: Analysis on time and error of polynomials with degree from 2 to 10 and distance between roots $d = 1$.

Polynomial degree	Roots expected	Roots computed	CPU time	relative error
2	1	0.9949	1.5515	0.0088
	2	1.9999		
3	1	0.9933	1.6619	0.0109
	2	2.0077		
	3	3.0133		
4	1	1.0075	1.8546	0.0101
	2	2.0105		
	3	2.9818		
	4	4.0049		
5	1	1.0047	1.8925	0.0093
	2	2.0059		
	3	3.0076		
	4	3.9790		
	5	5.0148		
6	1	1.0036	1.9306	0.0112
	2	1.9893		
	3	3.0078		
	4	4.0010		
	5	4.9951		
	6	6.0305		
7	1	1.0021	1.9529	0.0129
	2	2.0060		
	3	3.0045		
	4	4.0222		
	5	4.9865		
	6	5.9863		
	7	6.9821		
8	1	0.9918	1.9559	0.0120
	2	2.0040		
	3	2.9811		
	4	4.0228		
	5	4.9907		
	6	6.0067		
	7	7.0188		
	8	7.9659		
9	1	1.0063	2.0270	0.0117
	2	2.0059		
	3	2.9887		
	4	4.0420		
	5	4.9992		
	6	5.9731		
	7	7.0040		
	8	8.0119		
	9	8.9288		
10	1	1.0059	2.0413	0.0143
	2	2.0007		
	3	2.9517		
	4	3.9704		
	5	5.0589		
	6	5.9699		
	7	7.0388		
	8	8.0323		
	9	8.9915		
	10	10.0435		

TABLE II: Analysis of error of polynomials with root distance $d = 1, 2, 3, ..., 10$ for polynomials with degree 6.

Root distance	Roots expected	Roots computed	CPU time	relative error
2	1	0.9704	1.8136	0.0245
	3	3.2461		
	5	5.0472		
	7	6.9587		
	9	8.9861		
	11	11.0065		
3	1	0.9696	1.9683	0.0171
	4	4.0229		
	7	7.0049		
	10	10.0002		
	13	13.2351		
	16	16.0334		
4	1	1.3810	1.7809	0.0451
	5	5.3857		
	9	9.3837		
	13	13.4033		
	17	16.7586		
	21	20.9981		
5	1	0.9704	1.8129	0.0410
	6	7.0444		
	11	11.0069		
	16	15.9606		
	21	20.6787		
	36	25.9259		
6	1	0.9141	1.7743	0.0605
	7	8.1704		
	13	13.6354		
	19	19.9173		
	25	24.4222		
	31	30.7993		
7	1	1.0451	1.7853	0.0117
	8	7.9039		
	15	14.9219		
	22	22.0091		
	29	29.1785		
	36	36.1384		
8	1	0.9664	1.7756	0.0088
	9	9.0371		
	17	17.0713		
	25	24.9826		
	33	32.9873		
	41	41.0054		
9	1	1.1422	1.7684	0.0269
	10	9.9993		
	19	19.0200		
	28	28.1343		
	35	35.7041		
	46	46.0896		
10	1	0.9965	1.7587	0.0070
	11	10.9979		
	21	20.9666		
	31	31.0720		
	41	40.9753		
	51	51.1848		