Root finding and approximation approaches through neural networks

M.G. Epitropakis and M.N. Vrahatis

Computational Intelligence Laboratory, Department of Mathematics, University of Patras,
University of Patras Artificial Intelligence Research Center (UPAIRC), University of Patras,
GR–26110 Patras, Greece.
<mikeagn><vrahatis>@math.upatras.gr

Abstract

In this paper, we propose two approaches to approximate high order multivariate polynomials and to estimate the number of roots of high order univariate polynomials. We employ high order neural networks such as Ridge Polynomial Networks and Pi – Sigma Networks, respectively. To train the networks efficiently and effectively, we recommend the application of stochastic global optimization techniques. Finally, we propose a two step neural network based technique, to estimate the number of roots of a high order univariate polynomial.

1 Introduction

Two important scientific problems that arise in a large variety of applications are the approximation of multivariate high order polynomials, and the estimation of roots of high order univariate polynomials. Among the various proposed approaches for these problems Artificial Neural Networks have recently demonstrated their ability to provide accurate and fast approximations, as well as, root estimations. In this contribution we employ High Order Neural Networks, and more specifically Ridge Polynomial Networks, for the approximation of multivariate polynomial equations. We recommend the application of stochastic global optimization techniques, such as Differential Evolution and Particle Swarm Optimization, for the learning process in order to obtain better solutions. We further propose a two step technique for the estimation of a number of arbitrary roots of high order polynomials. This technique, at a first step, employs a trained feedforward neural network that acts as an oracle and returns the number of real roots of the univariate polynomial. At the next step, this information is used by a known high order neural network root-finder to estimate the roots.

The rest of the paper is organized as follows. Section 2 briefly describes two classes of high order neural networks, the Pi-Sigma and the Ridge polynomial networks. Section 3 is devoted to the exposition of the root-finding approach, while Section 4 presents the approximation approach. The paper ends with concluding remarks and a discussion in Section 5.

2 High Order Networks

In this paper, we consider the class of High Order Neural Networks and especially **Pi–Sigma Networks (PSN)**, and **Ridge Polynomial Networks (RPN)** that were introduced by Shin and Ghosh [9, 10]. Both networks have two basic types of neurons, namely "Pi" and "Sigma" neurons. Both types of neurons compute at a first step the product of each of their inputs with the corresponding weight. A "Sigma" neuron is a summing unit that computes the sum of the aforementioned products and subsequently applies an activation function on this sum. A "Pi" neuron, on the other hand, computes the product of the aforementioned products and applies an activation function on this. Both types of neurons can have an additional input called bias and its associated weight. Thus, a "Sigma" neuron returns $y = f(\sum_{i=1}^{n} x_i w_i - w_0)$, while a "Pi" neuron returns $y = f(\prod_{i=1}^{n} x_i w_i)$.

A **PSN** is a feedforward network that returns products of sums of input components. It consists of an input layer, a single hidden layer of summing units, and an output layer of product units. The weights connecting the input neurons to the neurons of the single hidden layer are adapted during the learning process, while those connecting the neurons of the hidden layer with the product units of the output layer are fixed (see Fig. 1 (a)). The degree of a PSN is the number of units in the hidden layer. Using a k-th degree PSN we obtain: $y(x_1, \dots, x_n) \simeq \sigma(\prod_{i=1}^{k} \sum_{j=1}^{n}(w_{ij}x_j + w_{i0}))$. The application of PSNs is motivated by their regular structure; their fast learning properties; and the fact that increasing the degree of the network does not require an exponential increase in the number of weights, as is the case for different types of networks [9, 10].

The **Ridge Polynomial Network**, [10] is a generalization of the PSN that employs a special form of ridge polynomials. A representation theorem is presented below that states that any multivariate polynomial can be represented in terms of a ridge polynomial, and can be realized by a suitable RPN (see Theorem 2.1 below) [9, 10].

RPNs have good mapping capabilities in the sense that any continuous function on a compact set in $\mathbb{R}^d$ can be uniformly approximated by such a network (see Theorem 2.2 below). RPNs are efficient in the sense that they utilize univariate polynomials which are easy to handle, in contrast to other high order networks that use multivariate polynomials, which causes an explosion of the number of weights (free parameters). Moreover, this type of network leads to a highly regular structure as compared to the ordinary higher–order networks, while maintaining its fast learning properties [9, 10].

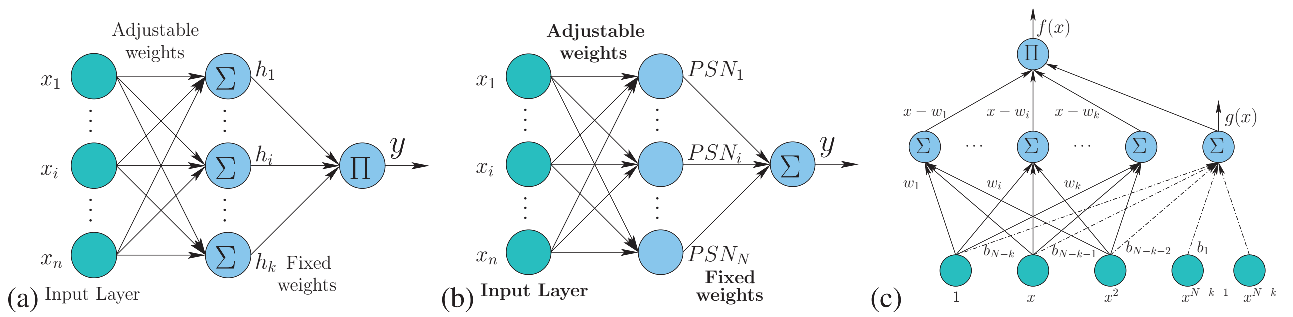

Figure 1: (a) Pi-Sigma Neural Network, (b) Ridge Polynomial Network, (c) Neural root-finder scheme.

Definition 2.1 *For a given compact set $K \subset \mathbb{R}^d$, all functions defined on K of the form, $f(\langle \cdot, \mathbf{w} \rangle) : K \mapsto \mathbb{R}$, where $\mathbf{w} \in \mathbb{R}^d$ and $f(\cdot) : \mathbb{R} \mapsto \mathbb{R}$ is continuous, are called **ridge functions**.*

A ridge polynomial is a ridge function that can be represented as: $\sum_{i=0}^{n} \sum_{j=1}^{m} a_{ij} \langle x, w_{ij} \rangle^i$.

Theorem 2.1 *Any multivariate polynomial can be represented as a ridge polynomial.*

$$p(x) = \sum_{j=0}^{k} \sum_{m=1}^{n_j} c_{jm} x^{i_{jm}} \Leftrightarrow p(x) = \sum_{j=i}^{N} \prod_{i=1}^{j} (\langle x, w_{ji} \rangle + w_{ji}).$$

Theorem 2.2 *Any continuous function on a compact set in $\mathbb{R}^d$ can be uniformly approximated by a Ridge Polynomial Network (a detailed description of the theorem and its proof can be found in [10]).*

From the above theorems it can be derived that, an RPN can be formed by the addition of PSNs of different degrees and this new structure has the same approximation capabilities with that of ordinary multivariate polynomials. Thus, an unknown function f defined on a compact set $K \subset \mathbb{R}^d$ can be approximated by an RPN as follows:

$$f(x) \simeq (\langle x, w_{11} \rangle + w_{11}) + (\langle x, w_{21} \rangle + w_{21}) \cdot (\langle x, w_{22} \rangle + w_{22}) + \cdots + (\langle x, w_{N1} \rangle + w_{N1}) \cdots (\langle x, w_{NN} \rangle + w_{NN}).$$

Where each product term is obtained as the output of a PSN with linear output units (see Fig. 1 (b)).

Given an RPN of degree up to k (PSN–degree) we obtain: $f(x) \simeq \sigma(\sum_{i=1}^{k} P_i(x))$, with $P_i(x) = \prod_{j=1}^{i}(\langle w_j, x\rangle + w_{j0})$, for $i = 1, \ldots, k$, where, $\sigma(\cdot)$ is a suitable activation function and the weights, w, are determined through a learning process.

3 Root finding Approach

Given a univariate high order polynomial we wish to estimate the real roots of $f(x)$

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

It is well known that numerous numerical methods are available to estimate a root of a polynomial, and using the deflation technique iteratively all the roots can be found one by one. Briefly, employing the deflation technique the next root is obtained by the deflated polynomial after the former root is found [1]. If we know that the polynomial $f(x)$ has k roots, we can factorize it as follows:

$$f(x) = (x - \rho_1)\cdots(x - \rho_k)g(x), \quad g(x) = x^{n-k} + b_1x^{n-k-1} + \cdots + b_{n-k-1}x + b_{n-k}.$$

The estimated roots thus obtained, are not guaranteed to be highly accurate because the rounding error can increase during the iterative process. Recently, methods have been proposed that exploit the parallel structure and the fast learning algorithms of neural networks to estimate all, or a number of roots accurately and in parallel [3, 4]. This kind of neural network is based on PSNs and it is called neural network root–finder. This scheme calculates simultaneously k real roots, and the coefficients of $g(x)$, as the values of the network weights (Fig. 1 (c))

A crucial parameter in the structure of this neural network is the number of roots that we wish to estimate, because as the order of the polynomial increases the number of the adjustable weights, that represent the roots, must also increase. The central idea of the proposed approach is to consult an "Oracle" that returns the number of real roots of the polynomial. As an "Oracle" we consider a trained feedforward neural network [6], that can return the number of real roots of a polynomial having as input the values of the coefficients. Compounding these we propose a technique that uses two components: An "Oracle" and a root-finding scheme.

The proposed technique consists of the following two steps: In the first step, we obtain from the "Oracle" the number of the real roots; let it be k. In the second step, we use k in the root-finder scheme in order to find rapidly and accurately the k real roots and the coefficients of $g(x)$.

4 Approximation approach

In this section we deal with the problem of approximating, or realizing efficiently and effectively, a multivariate high order function through RPNs. In particular, we consider a constructive learning algorithm and recommend the application of stochastic global optimization techniques, like Differential Evolution [11] and Particle Swarm Optimization [5], to train the Ridge Polynomial Network.

Due to the regular and systematic structure of RPNs each product term is obtained as the output of a PSN of degree i with linear output units. Therefore, the learning algorithms developed for PSNs can also be used for RPNs. An immediate approach is to select the optimal network structure for the function to be approximated and apply a learning algorithm. However, in general, the information required to determine the optimal network structure may not be a priori available. In these cases, instead of using a fixed network architecture, we apply the constructive learning method that evolves the architecture from a small network to a larger one, until the desired level of approximation error is attained. The main advantage of the constructive learning algorithm is that the networks realize a parsimonious approximation of the problem, typically resulting in better generalization [10]. For the learning process we propose the application of stochastic global optimization techniques, instead of traditional gradient descent, or Least Mean Squared algorithms, to enhance the learning process of the RPN.

4.1 Differential Evolution Algorithm and Particle Swarm Optimization

In this subsection, we briefly describe these global optimization algorithms. In a recent work, Storn and Price [11] presented a novel minimization method, called Differential Evolution (DE), designed to handle non–differentiable, nonlinear and multimodal objective functions. DE exploits a *population* of potential solutions, that is n–dimensional vectors, to probe the search space. At each iteration of the algorithm, called *generation*, three steps, *mutation*, *recombination* and *selection*, are performed in order to obtain more accurate approximations to a solution [8]. PSO, on the other hand, is a swarm–intelligence optimization algorithm [5]. Each member of the swarm, called *particle*, moves with an adaptable velocity within the search space, and retains in its memory the best position it ever encountered. In the *local* variant, each particle is assigned to a neighborhood consisting of a prespecified number of particles. At each iteration, the best position ever attained by the particles of the neighborhood is communicated among them [2]. In the *global* variant, the neighborhood of each particle is the entire swarm. In general, PSO and DE have proved to be efficient and effective in tackling numerous difficult real–world problems [7, 8].

5 Concluding Remarks and Discussion

This study presents two ideas. Firstly, a fast and accurate technique to find simultaneously a set of roots of a univariate polynomial is proposed. Secondly, an approach for approximating high order multivariate functions is presented. The root-finding approach can be easily parallelized because of the structure of the neural network. On the other hand, ridge polynomial networks are able to approximate high order functions efficiently and effectively [10]. They can be easily extended to a multidimensional classifier, and from the expansion of the trained network we can easily obtain the coefficients of the high order polynomial. Moreover, the use of stochastic global optimization techniques can enhance the learning process of an RPN. The performance of both approaches will be further investigated through extensive experimentation.

References

[1] K.M. Brown and W.B. Gearhart, Deflation techniques for the calculation of further solutions of a nonlinear system, *Numerische Mathematik*, **16** (1971), no. 4, 334–342.

[2] R.C. Eberhart, P. Simpson, and R. Dobbins, *Computational intelligence PC tools*, Academic Press, 1996.

[3] D.S. Huang and Z. Chi, Neural networks with problem decomposition for finding real roots of polynomials, *International Joint Conference on Neural Networks*, July 2001, pp. 25–30.

[4] D.S. Huang, H.H.S. Ip, and Z. Chi, A neural root finder of polynomials based on root moments, *Neural Computation*, **16** (2004), no. 8, 1721–1762.

[5] J. Kennedy and R.C. Eberhart, Particle swarm optimization, *Proceedings IEEE International Conference on Neural Networks*, vol. IV, IEEE Service Center, 1995, pp. 1942–1948.

[6] B. Mourrain, N.G. Pavlidis, D.K. Tasoulis, and M.N. Vrahatis, Determining the number of real roots of polynomials through neural networks, *Computers and Mathematics with Applications*, 2006, (in press).

[7] K.E. Parsopoulos and M.N. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, *Natural Computing*, **1** (2002), no. 2–3, 235–306.

[8] V.P. Plagianakos and M.N. Vrahatis, Parallel evolutionary training algorithms for 'hardware–friendly' neural networks, *Natural Computing*, **1** (2002), 307–322.

[9] Y. Shin and J. Ghosh, The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation, *International Joint Conference on Neural Networks*, vol. 1, July 1991, pp. 13–18.

[10] Y. Shin and J. Ghosh, Ridge polynomial networks, *IEEE Transactions on Neural Networks*, **6** (1995), 610–622.

[11] R. Storn and K. Price, Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces, *Journal of Global Optimization*, **11** (1997), 341–359.