DYNAMIC PROGRAMMING ALGORITHM FOR TRAINING FUNCTIONAL NETWORKS

Emad A. El-Sebakhy
Information and Computer Science Department
CCSE, KFUPM
Dhahran, Saudi Arabia, 31261
Email:sebakhy@kfupm.edu.sa

Salahadin A. Mohammed
ICS Department
CCSE, KFUPM
Dhahran, Saudi Arabia, 31261
Email:adam@kfupm.edu.sa

Moustafa A. Elshafei
Systems Engineering Department
CCSE, KFUPM
Dhahran, Saudi Arabia, 31261
Email:elshafei@ccse.kfupm.edu.sa

Abstract— **The paper proposes a dynamic programming algorithm for training of functional networks. The algorithm considers each node as a state. The problem is formulated as finding the sequence of states which minimizes the sum of the squared errors approximation. Each node is optimized with regard to its corresponding neural functions and its estimated neuron functions. The dynamic programming algorithm tries to find the best path from the final layer nodes to the input layer which minimizes an optimization criterion. Finally, in the pruning stage, the unused nodes are deleted. The output layer can be taken as a summation node using some linearly independent families, such as, polynomial, exponential, Fourier,...etc. The algorithm is demonstrated by two examples and compared with other common algorithms in both computer science and statistics communities.**

Keywords: Functional Networks; Dynamic Programming; Minimum Description Length; Interpolation.

I. INTRODUCTION

One of the strengths of the standard artificial neural network (ANN) is their capability to learn complex nonlinear relationships by training, which otherwise can hardly be modeled statistically or from first principles [1, 12]. Neural networks have been trained to perform complex functions in various fields of application including pattern recognition, identification, classification, prediction, speech recognition, computer vision, and control systems. ANNs are composed of signal processing elements called neurons. Each node consists basically of a summation node and an activation function. The neurons are interconnected and the strength of their interconnections is denoted by the parameters called synaptic weights. As it is shown in Figure1, the architecture of the Feed-Forward Neural Network (FFNN) is composed of layers of interconnected neurons. Usually, for sufficient number of hidden units, FFNNs can approximate any continuous static input-output mapping to any desired degree of approximation [2, 3]. The learning process involves updating network architecture and connection weights, so that a network can do both prediction and classification task using some efficient learning algorithms. The back propagation (BP) algorithm is usually used for FFNN training [2-5]. Several improvements of the BP algorithms were proposed in the literature, *e.g.* Delta-Bar-Delta algorithm [5, 8], Quick-propagation [6], and more recently the resilient back-propagation [7]. Several types of ANNs have also been proposed, such as radial basis function networks (RBFNs), and probabilistic neural networks (PNNs) [12]. Polynomial learning neural networks have been proposed for function approximation [11]. The concept stems from the fact that any continuous function can be approximated to an arbitrary precision in an average squared residual sense by multivariate Kolmogorov-Gabor polynomials. The advantage of polynomial learning networks is that they make possible the evaluation of complex, high-order models in acceptable time by composing simple transfer polynomials with weights that are computed efficiently by ordinary least-squares (OLS) fitting. Unlike Neural Networks, they can be trained with substantially smaller set of training data.

Recently, [9] proposed a method for constructing the polynomial in the form of a network of simpler two variable transfer polynomials. The selection of the transfer polynomials and construction of the network is performed using inductive genetic programming. Polynomial learning networks are also known as Abductive Networks. Abductive reasoning is defined as reasoning from general principles and initial facts to new facts under uncertainty [10, 12]. In an abductive network, complex systems are decomposed into smaller, simpler subsystems and grouped into several layers by using polynomial functional nodes. In the meantime, inputs are also subdivided into groups, then transmitted into individual functional nodes. These nodes evaluate the limited number of inputs by a polynomial function and generate an output to serve as an input to subsequent nodes of the next layer.

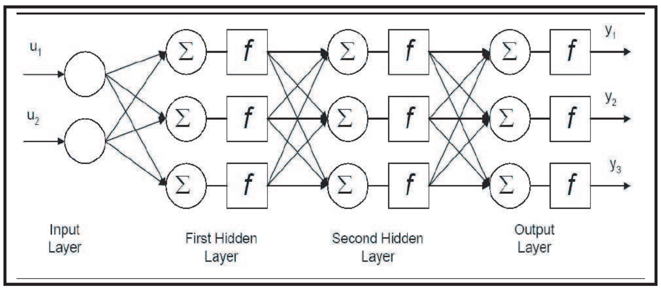

Fig. 1. Multi-layer feed forward neural network (FFNN).

Recently, functional networks Recently, functional networks have been introduced by [13], [14] as a generalization of the standard neural networks that combine both knowledge about the structure of the problem, to determine the architecture of the network, and data, to estimate the unknown functional neurons, see [15] for more details. The contribution of this paper is to propose a dynamic programming (DP) algorithm for training of functional networks, by considering each node as a state. Therefore, the problem is formulated as finding the sequence of states which minimizes the sum of the squared errors approximation. In addition, each node is optimized with regard to its corresponding neural functions and its estimated neuron functions. Furthermore, we find the best path from the final layer nodes to the input layer which minimizes an optimization criterion. The output layer can be taken as a summation node using some linearly independent families.

The rest of the paper is organized as follows: Section II provides a brief description of the state-of-the art of functional networks. Section III describes the polynomial learning scheme in details. The dynamic programming algorithm is proposed in Section IV. Distinct examples with implementation results are proposed in Section V. The conclusion and future work are presented in Section VI.

II. FUNCTIONAL NETWORKS: AN OVERVIEW

Functional networks are a generalization of neural networks that combine both knowledge about the structure of the problem, to determine the architecture of the network, and data, to estimate the unknown functional neurons [13], [14] and [15]. Functional networks as a new modeling scheme has been used in solving both prediction and classification problems. It is a general framework useful for solving a wide range of problems in probability, statistics, signal processing, pattern recognition, functions approximations, real-time flood forecasting, science, bioinformatics, medicine, structure engineering, and other business and engineering applications, see [15], and the references therein for more details. The performance of functional networks has shown bright outputs for future applications in both industry and academic research of science and engineering based on its reliable and efficient results. Several comparative studies have been carried to compare its performance with the performance of the most popular prediction/classification modeling data mining, machine learning schemes in literature [15]. The results show that functional networks performance outperforms most of these popular modeling schemes in machine learning, data mining, and statistics communities. Dealing with functional networks required some concepts and definitions, which can be briefly discussed as follows:

We call the node $X_j \in \mathbf{X}$, for all j as a *multiple node*, if it is an output of more than one neural functions. Otherwise, it is called a *simple node*.

As it is shown in Figure 2, a functional network consists of: a) several layers of storing units, one layer for containing the input data (x_i; $i = 1, \ldots, 4$), another for containing the output data (x_7) and none, one or several layers to store intermediate information (x_5 and x_6); b) one or several layers of processing units that evaluate a set of input values and delivers a set of

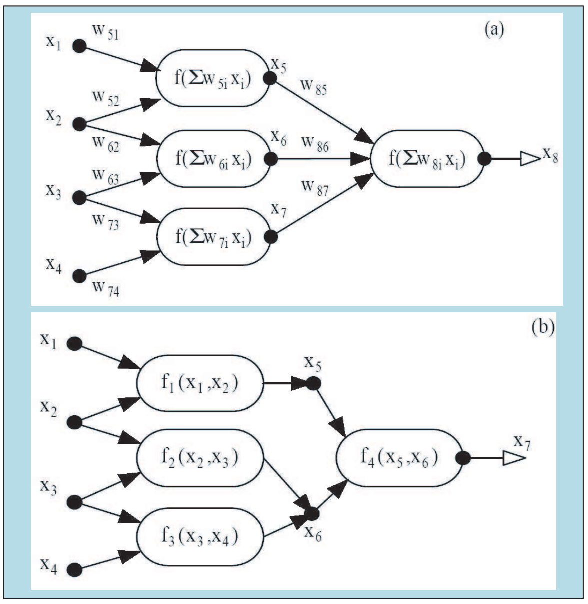

Fig. 2. (a) A Neural Network, (b) A Functional Network.

output values (f_i) and c) a set of directed links. Functional networks extend neural networks by allowing neural functions f_i to be not only true multi-argument and multivariate functions, but to be different and learnable, instead of fixed functions. In functional networks, the activation functions are unknown functions from a given family, *i.e.* polynomial, to be estimated during the learning process. In addition, functional networks allow connecting neuron outputs, forcing them to be coincident. Some differences between a neural network and a functional network are shown in Figure 2. Functional networks methodology can be more easily understood by organizing it into seven steps as it is shown in [15].

III. POLYNOMIAL LEARNING NETWORKS

We can write the output units in several distinct forms (one per different link) and get a system of functional equations that can be written directly from the architecture of the functional network. By solving this system of functional equations, we obtain an equivalent but simpler functional network [13, 14, 15]. We can learn the neuron functions by *structural learning* and/or *parametric learning*. In this paper we introduce a dynamic programming algorithm for selection of the transfer polynomials and optimization of the network parameters. To demonstrate the algorithm we consider the class of bivariate polynomial functions as shown in Table 1. We are interested in constructing a neural network where in the nodes can take a function from a predefined set of functions. Further more, the selection of the optimal function and the parameters of the functions are determined during the training the network. Clearly, the predefine set, F; can be expanded to include single node functions with a single input variable, and higher multivariate forms. The proposed structure of the Polynomial network as it is shown in Figure1 consists of one or more layers of transfer polynomials and an output layer.

The output layer could be simply a summation node. We want to investigate functional network structures consisting of T layers in which we have at most $l_1, l_2, \ldots, l_T$ nodes in the corresponding layers, and a single node output layer. The objective is to approximate an unknown function from a given set of training data. Extension of the algorithm to more general structures will be discussed as well.

Transfer Polynomials
$f_1(x) = w_0 + w_1 * x_1 + w_2 * x_2$
$f_2(x) = w_0 + w_1 * x_1 + w_2 * x_1 * x_2$
$f_3(x) = w_0 + w_1 * x_1 + w_2 * x_1 * x_2 + w_3 * x_1^2$
$f_4(x) = w_0 + w_1 * x_1 + w_2 * x_2^2$
$f_5(x) = w_0 + w_1 * x_1 + w_2 * x_2 + w_3 * x_1 * x_2 + w_4 * x_1^2 + w_5 * x_2^2$
$f_6(x) = w_0 + w_1 * x_1^2 + w_2 * x_2^2$
$f_7(x) = w_0 + w_1 * x_1 + w_2 * x_2 + w_3 * x_1^2 + w_4 * x_2^2$
$f_8(x) = w_0 + w_1 * x_1 + w_2 * x_1 * x_2 + w_3 * x_1^2 + w_4 * x_2^2$
$f_9(x) = w_0 + w_1 * x_1 * x_2$
$f_{10}(x) = w_0 + w_1 * x_2 + w_2 * x_1^2$

IV. DYNAMIC PROGRAMMING ALGORITHM

The dynamic programming algorithm considers each node as a state. The problem is formulated as finding the sequence of states which minimizes the sum of the squared of the approximation error. The dynamic programming algorithm tries to find the best path from the final layer nodes to the input layer which minimizes an optimization criterion. The minimization criterion is considered here the least squared error. At each layer, t, and for each node j, the algorithm search for the inputs from the previous layers which provide the best approximation of the output in the mean squared error sense. The parameters (weights) of the output layer are also obtained by *OLS* method. We assume we have N inputs $x_1, x_2, \ldots, x_N$, and a scalar output y. The training data consists of Minput vectors, and the size of the set of transfer polynomials is K.

We observed that to give a complete picture to the reader, we must define numerous notations, such as, $x_i(m)$ is the m^{th} sample of the i^{th} input variable. In addition, $v_{t,i}(m)$ is the $m^t h$ sample of output of i^{th} node in the t^{th} layer. Furthermore, layer $E(t,i)$ is *LSE* of the i^{th} node in the t^{th} layer and $\mathbf{D}$ is an $N \times M$ matrix of the training data, and $\mathbf{Y}$ is an $M \times 1$ vector of the desired output. The details of the entire training algorithm steps are expressed as follows:

1) At the input layer, for each pair of input variables we select the best transfer polynomial which fits the given data in the least squared error sense.

$$J(i,j) = \begin{matrix}\min \\ k, w\end{matrix} \{\sum_{m=1}^{M} (y(m) - f_k(w, x_i(m), x_j(m)))^2\} \tag{1}$$

for $i, j = 1, 2, \ldots, N$. The polynomial is defined by its index k, and its coefficient vector $\mathbf{W}$. For each k, we apply the OLS technique to find the parameter vector $\mathbf{W}$, and compute the residual mean squared error. The best polynomial is the one which achieves the least residual sum of squared error. Clearly, the number the generated nodes is N^2

2) Next, we sort the nodes in ascending order and select the best l_1 nodes, where l_1 is the desired number of nodes in the input layer.
3) Compute the outputs $\{v_j^1(m)\}$,for$j = 1, 2, ...l_1$.
4) The nodes of the subsequent layers are computed by repeating the same steps, however by replacing the input x by v, i.e.

$$J(i,j) = \begin{matrix}\min \\ k, w\end{matrix} \{\sum_{m=1}^{M} (y(m) - f_k(w, v_i(m), v_j(m)))^2\} \tag{2}$$

The residual minimum error is then sorted and the best l_i node is retained. The output from the retained nodes are computed and used in the subsequent layers.

5) Pruning: starting from the T layer, that is the final layer excluding the output layer, we trace back the nodes, identifying the signal path from each node in the final layer to the input variables. We only keep the nodes along these paths, while the unused nodes are deleted. The network in this case will have at most l_1 nodes in the firs layer, at most l_2 nodes in the second layer, and so on.
6) Finally the weights of the output node is computed by minimizing the squared error

$$E = \begin{matrix}\min \\ w\end{matrix} \{\sum_{m=1}^{M} (y(m) - \sum_{l=1}^{Lf} w_{f,l} v_l^f(m))^2\}, \tag{3}$$

using the ordinary least squares (OLS) techniques. The use of summation node rather than a function avoids over fitting the data. The algorithm can easily be extended to allow for inclusion of inputs from any of the previous layers by including an identity function, and other single node functions in the predefined set of functions.

V. EXAMPLES

Example 1: Pollutant in the Exhaust Gas of a Boiler

The concentration of a pollutant in the exhaust gas of a boiler is measured under various operating conditions. The operating conditions are determined by 6 variables: Air fuel ratio, Fuel mass flow rate, Air flow rate, Flame maximum temperature, Average combustion chamber temperature, and the outlet gas temperature.

The training data consists of 10 sets of readings only. While the test data consists of two sets of readings. A polynomial learning network was constructed using 12 nodes in the first layer and 4 nodes in the second layer. The trained network achieved the following *network (net)* values for the training data. The predicted values and the actual values for the testing data are summarized. The predicted versus the actual values for the test data

Network	53.39	58.37
Actual concentration (ppm)	53.5	58.1

The structure of the nodes of the first layer is summarized in the following table:

Node	First input	Second input	Trans. poly.
1	4	1	7
2	4	2	5
3	1	2	5
4	3	2	5
5	6	3	5
6	5	4	5
7	3	1	5
8	6	4	5
9	5	1	5
10	5	3	5
11	2	5	5
12	6	2	5

The second layer consists of four nodes, that is, as follows

Node	First input	Second input	Trans. poly.
1	7	11	5
2	3	11	5
3	2	10	5
4	6	1	5

The second layer doe not utilize all the nodes of the first layer, and hence these unused nodes of the first layer, namely $(4, 5, 8, 9, 12)$ are deleted from the first layer.

Example 2: Non Polynomial Function

Suppose that we generate data of size 200 from the model $\log(y) = x_1 + x_2 + \varepsilon$, where $x_1 \sim U[0, 1]$, $x_2 \sim U[0, 1]$, and $\varepsilon \sim N[0, 0.01]$ or $\varepsilon \sim N[0, 0.1]$. The scatter plot of $\mathbf{y}$ against $x_1 + x_2$ is shown in Figure3 with sample of 50 observations for both models. To determine the estimated model using functional networks using the proposed training algorithm with separable functional network model, we choose the polynomial families of linearly independent functions $\phi_1 = \{1, y, y^2\}$, $\phi_2 = \{1, x_1, x_1^2, x_1^3\}$, $\phi_3 = \{1, x_2, x_2^2, x_2^3, x_2^4\}$, and tolerance values equal to 0.1 or 1, see [14, 15] for more details. Thus, by applying the dynamical programming algorithm on the initial architecture of the corresponding functional network [14], we obtain the estimated functional network model with $\varepsilon \sim N[0, 0.01]$ with backward selection criterion is given by:

$$\begin{aligned} &-0.9498 + 1.193\,y - 0.2071\,y^2 + 0.0141y^3 \\ &= 0.138\,x_1 - 0.1532\,x_1^3 + x_2. \end{aligned} \tag{4}$$

with the value 6.72×10^{-6} for the estimated *mean squared errors* (MSE).

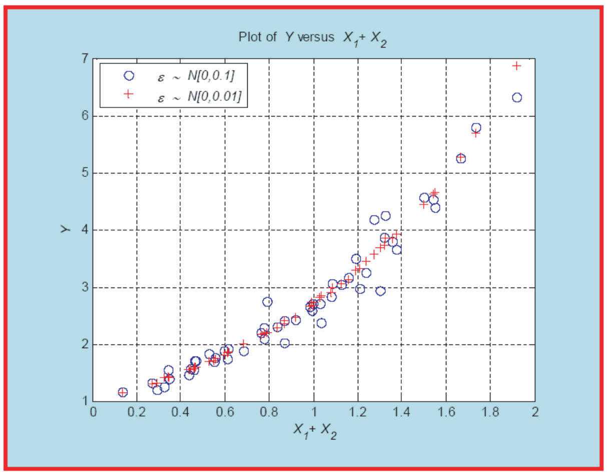

Fig. 3. Scatter plots of the genearated data using given models with both $\varepsilon \sim N[0, 0.1]$ and $\varepsilon \sim N[0, 0.01]$.

When data are simulated without error (no uncertainty associated with the data sample), the true functional network model is selected at the end of backward procedure if tolerance =1. The given logarithmic model is approximated by functional networks with polynomial function, that is,

$$-0.8280 + 1.1930\,y - 0.2071\,y^2 + 0.0141\,y^3 = x_1 + x_2.$$

Figure 4 shows the scatter plots of both natural logarithm $(ln(y))$ and the predicted model: $-0.8280 + 1.1930\,y - 0.2071\,y^2 + 0.0141\,y^3$ against $x_1 + x_2$. We note that both graphs are identical.

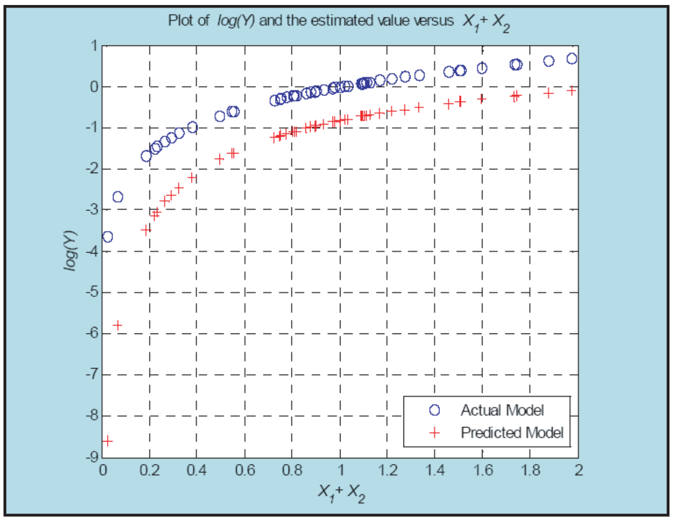

Fig. 4. Plots of the actual and predicted models versus $x_1 + x_2$.

We observed that the estimated form of the functional networks is simpler than its given model. In addition, the output estimated value layer doe not utilize all the input nodes. This is reasonable because more different terms are considered in these models, then tolerance value influence very much in the results.

VI. Conclusion

The paper provided a dynamic programming algorithm to train polynomial learning networks. The algorithm can be applied to any functional networks architectures and produce both reliable and efficient performance . The algorithm is suitable for limited training data, and discovers automatically the best structure for the network. For future work we suggest more simulation study and real-world applications to be investigated and carry comparative studies with the most popular schemes in both data mining and machine learning communities.

Acknowledgment

The authors wish to acknowledge the support and facilities provided by King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia in the development of this paper.

References

[1] Rumelhart D.E., Hinton G.E., and Williams R.J.,Learning internal representation by back Propagation. *In Parallel Distributed Processing: Exploration in the Microstructure of Cognition, V. I, Cambridge, MA:*, (1986).

[2] Simon H., Neural Networks: a Comprehensive Foundation, *Macmillan Publishing Englewood Cliffs, NJ,* (1994).

[3] Stinchcombe M. and White H., *Multilayer feed-forward network are universal approximators*. Neural Networks, v.2, p 359–366, (1989).

[4] Hopfield J. J., *Neural networks and physical systems with emergent collective computational abilities*. Proc. Of the National Academy of Sciences of the U.S.A., Vol. 79, pp. 2554–2558, (1982).

[5] Jacob R., *Increased Rates of Convergence Through Learning Rate Adaptation*. Neural Networks, Vol. 1, pp. 295–307, (1988).

[6] Fahlman S.E., *An Empirical Study of Learning Speed in Back propagation Networks*. Technical report, CMU-CS-88–162, Carngie-Mellon University, (1988).

[7] Riedmiller R. and Braun H., *A Direct Adaptive Method for Faster Back Propagation Learning: The Prop Algorithm*. Proc. of the IEEE International Conference on Neural Networks, pp. 586–591, (1993).

[8] Fernandez B., Parlos A. G., and Tsai W. K., *Nonlinear Dynamic System Identification Using Artificial Neural Networks*. Proc. of International Joint Conference On Neural Networks, Vol. 2, pp. 131–141, (1990).

[9] Nikolaev N. and Iba H. *Learning Polynomial Feedforward Neural Networks by Inductive Genetic Programming and Backpropagation*, IEEE Transactions on Neural Networks, vol.14, N:2, pp.337–350, (2003).

[10] Lee Y. B., Liu H. S., and Tarng Y. S. *An Abductive Network for Predicting Tool Life in Drilling*. IEEE Transactions on industry applications, vol. 35, No. 1, (1999).

[11] Hema R., Madala A. Grigoévich I., *Inductive learning algorithms for complex systems modeling*. CRC Press, © (1994).

[12] Duda R. O., Hart P. E., and Stock D. G., *Pattern Classification*, Second Edition, John Wiley & sons, NY, (2000).

[13] Castillo E. and Ruiz-Cobo R., *Functional Equations and Modelling in Science and Engineering*, Marcel Dekkerm NY, (1992).

[14] Castillo E., Cobo A., Gutiérrez J. M. and Pruneda E., *Introduction to Functional Networks with Applications, A Neural Based Paradigm*, Kluwer Academic Publishers: New York, (1998)

[15] El-Sebakhy A. E., Hadi S. A., and Faisal A. K., Iterative Least Squares Functional Networks Classifier; *IEEE Transactions Neural Networks*, vol. 18, no. 2, March (2007).