A neural feedforward network with a polynomial nonlinearity

Hoffmann, Nils

Published in:
Proceedings of the IEEE-SP Workshop Neural Networks for Signal Processing

Link to article, DOI:
10.1109/NNSP.1992.253708

Publication date:
1992

Document Version
Publisher's PDF, also known as Version of record

A NEURAL FEED-FORWARD NETWORK WITH A POLYNOMIAL NONLINEARITY

Nils Hoffmann
Electronics Institute, Building 349
Technical University of Denmark
DK-2800 Lyngby, Denmark
Phone: +45 45931222 ext. 3916
Fax: +45 42880117

INTRODUCTION

The problem of nonlinear, adaptive filtering has been dealt with using many different filter architectures among which we find the neural feed-forward network [4] and the Volterra filter [6]. In this paper a synthesis of these two architectures is proposed in the form of a new feed-forward net designed for the purpose of nonlinear filtering of time series. The general nonlinear filter

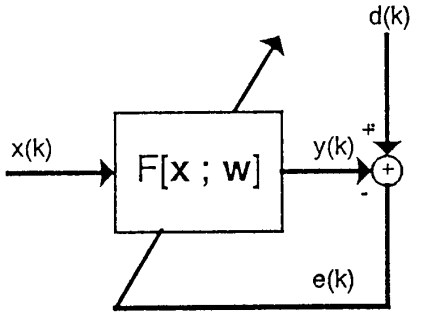

Figure 1: Nonlinear adaptive filtering configuration. $x(k)$ is the input, $y(k)$ the output, and $d(k)$ the desired signal. The filter is adapted in order to minimize the mean square error, $\sum_k e^2(k)$.

configuration is shown in Fig. 1. This configuration may perform tasks such as prediction and identification of transfer functions [9]. The filter is designed to implement the equation:

$$y(k) = F[x(k), x(k-1), \cdots, x(k-L+1); \mathbf{w}] \tag{1}$$

where $F[\cdot]$ is an unknown nonlinear function parameterized by $\mathbf{w}$, k is the discrete time index, and L is the filter order. Note, that we have restricted

0-7803-0557-4/92$03.00 © 1992

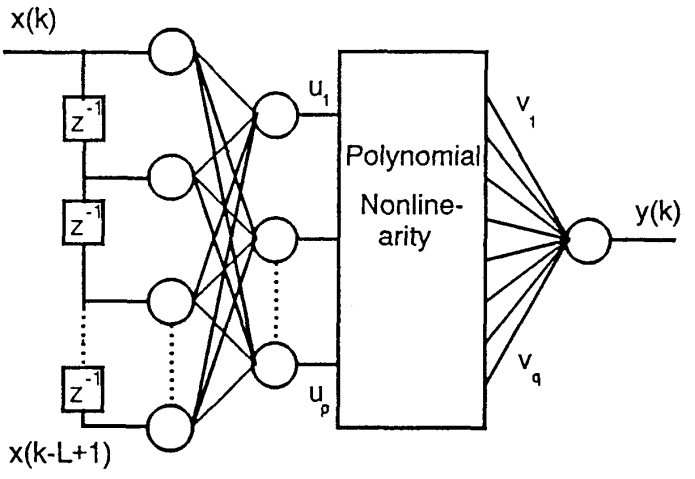

Figure 2: Network architecture.

the nonlinear mapping of $x(k)$ into $y(k)$ to be non-recursive which ensures the BIBO-stability of the filter. In the following sections a description of the new network architecture and algorithms for estimating the network parameters will be given along with an analysis of the computational complexity. Finally, we show a simulation in order to substantiate the usefullness of the proposed approach to nonlinear filtering.

NETWORK ARCHITECTURE

The proposed network architecture is shown in Fig. 2. The network, which may be viewed as a generalization of the Wiener model [6], is composed of three sections: A preprocessing unit consisting of L input neurons and p hidden neurons , a memoryless, multidimensional nonlinearity, and a linear output neuron. The main objective of the preprocessing is to extract the essential information contained in $\mathbf{x} = \{x_j\} = [x(k), x(k-1), \cdots, x(k-L+1)]^T$ without loosing information concerning $d(k)$ and simultaneously ensuring that $\mathbf{u}$ has a dimension, $p \leq L$. The polynomial nonlinearity is memoryless and transforms the vector $\mathbf{u}$ into the vector $\mathbf{v}$ of dimension $q = C_{m+p,m}$, where $C_{n,k}$ denotes the binomial coefficient, and m is the maximum order of the polynomial terms v_r, $r = 1, \cdots, q$. Finally the linear neuron forms a weighted sum $y(k)$ of the terms in $\mathbf{v}$. It is worth noting that for $m = 1$ the network is equal to a standard 2-layer feed-forward net. This implies a trade off between the complexity of the hidden layer and the complexity of the output neuron expressed in terms of p, m and q. The merit of the proposed architecture is that it shifts the complexity from the hidden neurons to the

linear output neuron, which is normally easier to adapt (fast convergence, no local minima, see e.g. [9], [1]).

Preprocessing network

The hidden nodes in the preprocessing network follow the equations:

$$u_i = \tanh(z_i), \quad z_i = \sum_{j=1}^{L} c_{ij} x_j, \quad i = 1, 2, \cdots, p \tag{2}$$

Where $x_j = x(k-j+1)$. In an earlier work [2] it was suggested to estimate the parameters c_{ij} solely on the basis of the input signal $x(k)$ using principal component analysis (PCA) [3]. This aproach has the obvious flaw of not using the desired signal in the calculation of c_{ij} which may lead to a significant loss of information concerning the mapping of $x(k)$ into $d(k)$ [2]. A way of overcoming this deficiency is to make c_{ij} adaptable and thus dependent on the overall performance criterion (i.e. the mean square error). This can be done using a variant of the well-known backpropagation algorithm [5]. However, it is a prerequisite that we develop a scheme for the efficient computation of the derivatives of the polynomial nonlinearity.

Polynomial nonlinearity

In accordance with the Wiener model [6] the nonlinearity combined with the linear output neuron is chosen such that $y(k)$ is expressed as a truncated polynomial expansion:

$$\begin{aligned} y(k) &= a_0 + \sum_{i_1=1}^{p} a_{1,i_1} P_1(u_{i_1}) + \sum_{i_1=1}^{p} \sum_{i_2=i_1+1}^{p} a_{2,i_1,i_2} P_1(u_{i_1}) P_1(u_{i_2}) \\ &\quad + \sum_{i_1=1}^{p} a_{2,i_1} P_2(u_{i_1}) + \cdots + \sum_{i_1=1}^{p} a_{m,i_1} P_m(u_{i_1}) \end{aligned} \tag{3}$$

Where $P_s(u_i)$ is a polynomial in u_i of order s and $a_{s,i_1,i_2,\cdots}$ are the weights in the output neuron. In this paper we will use a slightly different notation:

$$y(k) = \sum_{r=1}^{q} a_r v_r, \quad v_r = P_{s_1}(u_1) P_{s_2}(u_2) \cdots P_{s_p}(u_p), \quad \sum_{i=1}^{p} s_i \le m, \quad s_i \ge 0 \tag{4}$$

$\mathbf{v} = \{v_r\}$ contains *all* distinct products of the polynomials $P_{s_i}(u_i)$. If $P_s(u) = u^s$ Eq. (4) becomes a Volterra expansion. In the Wiener model it is assumed that the u_i's are independent, Gaussian variables. This assumption leads to the choice of using the Hermite polynomials as the product terms, v_r, in Eq. (4) consequently become orthogonal. Furthermore this entails that convergence in mean is assured for all values of u_i [6]. In general the distribution

of **u** is unknown which makes it impossible to find a set of orthogonal polynomials. However, due to the squashing functions in the hidden layer u_i is limited to the interval $]-1;1[$. This makes the Chebyshev polynomials an attractive choice as they limit v_r to $]-1;1[$. Furthermore they tend to be less correlated than the Volterra polynomials for most practical input distributions, partly due to the fact that the Volterra polynomials of even order contain a substantial mean value. This decreases the speed of convergence when adapting the output neuron using backpropagation. An alternative way of avoiding the problems of slow convergence for highly correlated v_r is to adapt the output neuron with a 2nd order algorithm of the Gauss-Newton type [7, Ch. 14].

The Chebyshev polynomials $T_s(u_i)$ and their derivatives $dT_s(u_i)/du_i$ may be obtained using the recurrence formulas:

$$T_{s+1}(u) = 2uT_s(u) - T_{s-1}(u), \qquad T_0(u) = 1, \;\; T_1(u) = u \tag{5}$$

$$\frac{dT_{s+1}(u)}{du} = 2T_s(u) + 2u\frac{dT_s(u)}{du} - \frac{dT_{s-1}(u)}{du}, \qquad \frac{dT_1(u)}{du} = 1, \;\; \frac{dT_2(u)}{du} = 4u \tag{6}$$

An efficient algorithm for calculation of the output terms v_r from the inputs u_i of the polynomial nonlinearity along with the derivatives $\partial v_r/\partial u_i$ is not easily derived from Eq. (4). Instead it is proposed to picture the computations as a tree traversing procedure as shown in Fig. 3. The tree traversing is done as follows: Initially the value 1 is in the accumulator (this term is used to denote a temporary variable) and as we move downwards we multiply with the polynomials marked on the branches we follow. The tree is divided into layers at depths $1 \cdots p$ such that the first branch we traverse is a polynomial in u_1 of order $s_1 \leq m$, the next branch a polynomial in u_2 of order $s_2 \leq m - s_1$ and so forth. At the terminal node the accumulator will be equal to an output term v_r. Looking at the trees it should be easy to realize, that the derivatives $\partial v_r/\partial u_i$ can be calculated using the same procedure. The only modification necessary is to replace the polynomials $P_s(u_i)$ with the corresponding derivatives $dP_s(u_i)/du_i$, $s = 0, 1, \cdots, m$. That is, in order to calculate both the polynomial terms v_r and their derivatives $\partial v_r/\partial u_i$ it is necessary to traverse one polynomial tree and p derivative trees. The pseudo code for the tree traversing procedure is very simple as shown below (Note that the code needs some (immaterial) modifications when implemented with variable p):

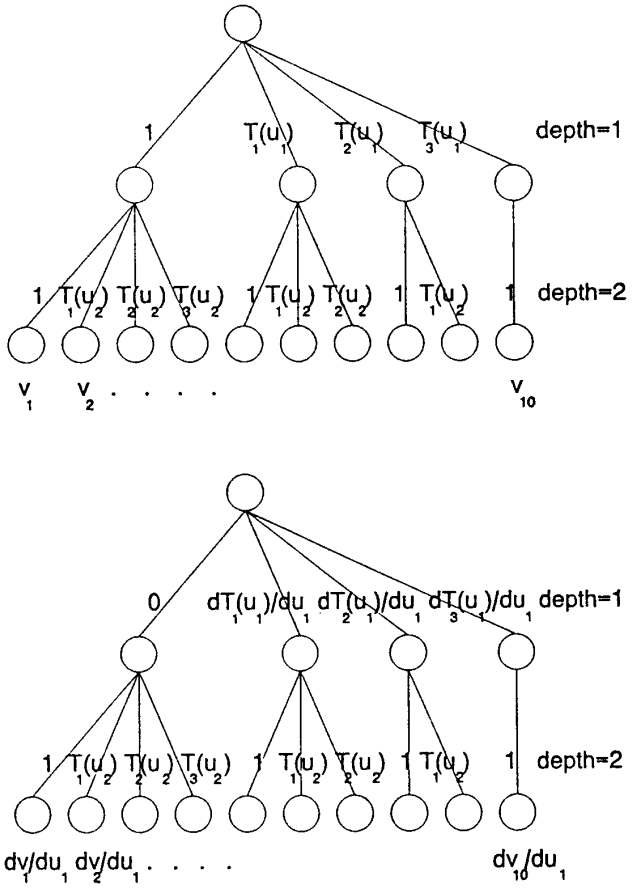

Figure 3: Calculation trees for calculating the polynomial terms and their derivatives. Maximum order $m = 3$, no. of input variables $p = 2$, and no. of output variables $q = 10$.

```
acc = 1
r = 1
for s_1 = 0, 1, ..., m
    acc = T_{s_1}(u_1) · acc
    for s_2 = 0, 1, ..., m − s_1
        acc = T_{s_2}(u_2) · acc
        for s_3 = 0, 1, ..., m − s_1 − s_2
            ...
                for s_p = 0, 1, ..., m − Σ_{n=1}^{p−1} s_n
                    v_r = T_{s_p}(u_p) · acc
                    r = r + 1
```

WEIGHT ESTIMATION ALGORITHM

The weights in the hidden neurons as well as in the output neuron may readily be estimated using a modified α-LMS algorithm [8]:

$$\Delta \mathbf{w} = \alpha e \frac{\frac{\partial y}{\partial \mathbf{w}}}{\left|\frac{\partial y}{\partial \mathbf{w}}\right|^2}, \quad \mathbf{w} = [c_{11}, \cdots c_{pL}, a_1, \cdots a_q]^T \tag{7}$$

$\Delta \mathbf{w} = \mathbf{w}(k+1) - \mathbf{w}(k)$ is the change in the weights due to updating and α is the normalized step-size. The algorithm remains stable for $\alpha \in [0;2[$, however selecting $\alpha > 1$ is not sensible [8]. Time index is omitted as all terms are evaluated at time k. The modification consists of replacing the input vector with the gradient $\partial y/\partial \mathbf{w}$ thus making the algorithm applicable to nonlinear systems. The partial derivatives are easy to find using the chain rule:

$$\frac{\partial y}{\partial c_{ij}} = x_j \frac{\partial u_i}{\partial z_i} \sum_{r=1}^{q} a_r \frac{\partial v_r}{\partial u_i} = x_j g_i, \quad \frac{\partial y}{\partial a_r} = v_r \tag{8}$$

This yields the following update equations:

$$\left|\frac{\partial y}{\partial \mathbf{w}}\right|^2 = \sum_{r=1}^{q} v_r^2 + \sum_{i=1}^{p} g_i^2 \cdot \sum_{j=1}^{L} x_j^2, \quad g_i = (1-u_i^2) \sum_{r=1}^{q} a_r \frac{\partial v_r}{\partial u_i} \tag{9}$$

$$\Delta a_r = \alpha e \frac{v_r}{\left|\frac{\partial y}{\partial \mathbf{w}}\right|^2}, \quad r = 1, \cdots, q \tag{10}$$

$$\Delta c_{ij} = \alpha e \frac{x_j g_i}{\left|\frac{\partial y}{\partial \mathbf{w}}\right|^2}, \quad i = 1, \cdots, p \quad j = 1, \cdots, L \tag{11}$$

COMPLEXITY

In order to properly evaluate the proposed network and weight estimation algorithm it is necessary to obtain an estimate of the computational complexity e.g. in terms of the number of multiplications/divisions needed in one

iteration of filtering and adaptation. For this purpose we use the calculation trees in Fig. 3 and state certain facts:

- There is $C_{m+n,m} = \begin{pmatrix} m+n \\ m \end{pmatrix}$ nodes at depth n in the tree, $n \in \{1, \cdots, p\}$ [6].
- The total number of multiplications in the tree is equal to the number of nodes minus 1.
- Calculation of p Chebyshev polynomials up to order m requires $p(m-1)$ effective multiplications (cf. Eq. 5) (i.e. multiplications with 1 and 2 are not counted).
- Calculation of p derivatives of Chebyshev polynomials up to order m requires $p(m-2)$ effective multiplications (i.e. multiplications with 1, 2 and 4 are not counted). (cf. Eq. 6).

From Fig. 3 it should be evident that the number of multiplications with 1 going from depth n to depth $n+1$ in the polynomial tree is equal to the number of nodes at depth n. The total number of times the accumulator is multiplied by 1 is thus equal to the total number of nodes minus the number of nodes in the final layer ($n = p$). The number of times where the accumulator is equal to 1 and is multiplied by a polynomial $P_s(u_i) \neq 1$ is mp. Using all these facts it follows that the number of effective multiplications is:

$$\begin{aligned} N_P &= \begin{pmatrix} m+p \\ m \end{pmatrix} - 1 - mp + p(m-1) \\ &= \begin{pmatrix} m+p \\ m \end{pmatrix} - (p+1) = q - (p+1), \quad m \geq 1, \ p \geq 1 \end{aligned} \tag{12}$$

where q is the dimension of $\mathbf{v}$. It is easy to show, that N_P reaches the lower bound on the number of multiplications necessary to calculate $\mathbf{v}$. Using similar arguments it can be shown that the number of effective calculations in the derivative tree is (The derivatives are simply polynomials of order $m-1$):

$$\begin{aligned} N_D &= \begin{pmatrix} m+p-1 \\ m-1 \end{pmatrix} - (p+1) \\ &= q\frac{m}{m+p} - (p+1), \quad m \geq 2, \ p \geq 1 \end{aligned} \tag{13}$$

Counting the number N_N of multiplications in the adaptation and filtering by the neurons is straight forward keeping in mind, that $\partial v_r / \partial u_i = 0$ in a fraction of $p/(m+p)$ times.

$$N_N = \left(\frac{mp}{m+p} + 3\right) q + 2(p+1)(L+1) - L + 3, \quad m \geq 2, \ p \geq 1 \tag{14}$$

The total number of multiplications per iteration turns out to be:

$$\begin{aligned} N &= N_P + pN_D + N_N \\ &= \left(2\frac{mp}{m+p} + 4\right) q + 2(p+1)(L+1) - (p+1)^2 - L + 3, \\ & \quad m \geq 2, \quad p \geq 1 \end{aligned} \tag{15}$$

Normally (cf. Section) q is much larger than L and p. That is, for most m $q\min(m,p) \gg pL$, in which case $(2mp/(m+p)+4)q$ is a rough estimate of the number of multiplications. This emphasizes the main difficulty with this kind of network, which is the fast growth of q with growing m and p. If the estimation of the weights in the hidden neurons is performed using PCA the complexity is roughly $4q$ multiplications per iteration, i.e. the relative change in complexity when implementing the backpropagation is:

$$\left(4 + \frac{2mp}{m+p}\right) q(4q)^{-1} \leq 1 + \frac{1}{2}\min(m,p) \tag{16}$$

SIMULATIONS

In order to illustrate the benefits of the proposed method of preprocessing compared to the PCA method, we have simulated a simple example of system identification. The unknown system is a network as shown in Fig. 2 with parameters

$$\mathbf{C} = \{c_{ij}\} = \begin{bmatrix} 0.3 & -0.25 & 0.1 & -0.15 \\ -0.2 & -0.1 & 0.2 & -0.1 \end{bmatrix} \tag{17}$$

$$\mathbf{a} = \{a_i\} = \begin{bmatrix} 0.2 & 0.5 & -0.4 & 0.7 & 0.6 & -0.8 \end{bmatrix}^T \tag{18}$$

Implying that $p = 2$, $L = 4$, $m = 2$, and $q = C_{2+2,2} = 6$. The input signal $x(k)$ is a colored sequence which is obtained by filtering zero mean white Gaussian noise with a 2nd order butterworth lowpass filter (cutoff at $0.15f_s$, where f_s denotes the sampling frequency). The two networks used for the identification were given the same architecture as the unknown system (i.e. $p = 2$, $L = 4$, $m = 2$), and the normalized step-size α was set to 0.3. A set of 200 samples was used to train the networks, and after each pass of the entire set the performance index $E = \sigma_e/\sigma_d$ was calculated. The standard deviations σ_e, σ_d of the cross-validation error and the desired signal were estimated using an independent set of 200 samples. Fig. 4 shows how the convergence of E depends on the choice of preprocessing method. The network using PCA displays a fast convergence but at the expense of a high final error index. Note that this network is linear in the parameters i.e. there are no local minima, and consequently there would be no significant decrease of E even if the number of passes was increased. The proposed net converges slower but eventually it reaches the optimal solution ($E = 0$, not shown in Fig. 4) unless the weight estimation algorithm is caught up in a local minimum.

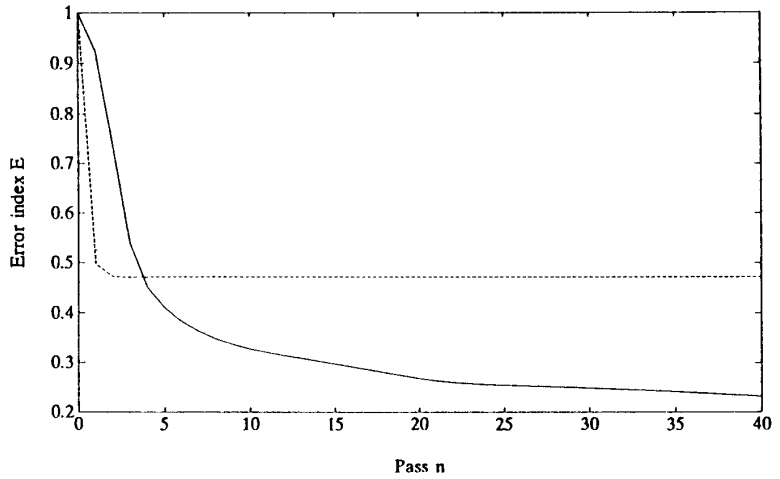

Figure 4: Convergence of the error index E as a function of the pass number n. Dashed line: The network using PCA. Solid line: The proposed network.

CONCLUSION

In this paper a new neural network based on the Wiener model has been proposed. The network is composed of a hidden layer of preprocessing neurons followed by a polynomial nonlinearity and a linear output neuron. In contrast to earlier suggestions [2] we try to solve the problem of finding an appropriate preprocessing method by using a modified backpropagation algorithm. It is shown by the use of calculation trees, that the proposed approach is simple to implement, and that the computational complexity is not much larger than for the alternative method of using PCA to determine the weights in the preprocessing network. Finally, a simulation has been given which indicates superior performance of the proposed network compared to the PCA network.

ACKNOWLEDGEMENTS

I would like to thank Lars Kai Hansen, Jan Larsen, Peter Koefoed Møller, and John Aasted Sørensen for helpfull comments on this paper.

REFERENCES

[1] S. Haykin, Adaptive filter theory, Englewood Cliffs, N.J.: PRENTICE-HALL, 1991.

[2] N. Hoffmann & J. Larsen, "A Neural Architecture for Nonlinear Adaptive Filtering of Time Series" in B.H. Juang, S.Y. Kung & C.A. Kamm (eds.)

Neural Networks for Signal Processing, Proceedings of the 1991 IEEE Workshop, Piscataway, N.J.: IEEE, 1991.

[3] P.R. Krishnaiah (ed.), Multivariate Analysis 2, New York, N.Y.: ACADEMIC PRESS, 1969.

[4] A.S. Lapedes & R. Farber, "Nonlinear Signal Processing Using Neural Networks, Prediction and System Modeling," Technical Report LA-UR-87, Los Alamos National Laboratory, 1987.

[5] J.L. McClelland & D.E. Rumelhart (eds.), Parallel Distributed Processing, Explorations in the Microstructure of Cognition. Vol. 1: Foundations, Cambridge, Massachusetts: MIT PRESS, 1986.

[6] M. Schetzen, The Volterra and Wiener Theories of Nonlinear Systems, Malabar, Florida: ROBERT E. KRIEGER PUBLISHING COMPANY, 1989.

[7] G.A.F Seber & C.J. Wild, Nonlinear Regression, New York, N.Y.: JOHN WILEY & SONS, 1989.

[8] B. Widrow, R.G. Winther & R.A. Baxter, "Layered Neural Nets for Pattern Recognition," IEEE Transactions on Acoustics Speech and Signal Processing, vol. 36, July 1988.

[9] B. Widrow & S.D. Stearns, Adaptive Signal Processing, Englewood Cliffs, N.J.: PRENTICE-HALL, 1985.