FEED FORWARD NEURAL NETWORKS FOR AERODYNAMIC MODELLING AND SENSOR FAILURE DETECTION

J.R Raol*

Abstract

In this paper applications of feed forward neural networks for aerodynamic modelling and sensor failure detection are studied. These networks are trained using fast back-propagation recursive least squares algorithms with forgetting factors. Specifically three training algorithms are implemented in PC MATLAB. The numerical results are presented using simulation of aircraft mathematical models.

Introduction

Recently the field of artificial neural networks (ANNs) has fascinated scientists and engineers all over the world. ANN modeling has become very popular in the area of signal processing, pattern recognition, system identification and control [1,2]. Neural networks have good ability to learn adaptively. In general the decision process in ANNs is based on inherently nonlinear operations. Such nonlinearities might be useful: to improve the convergence speed, to provide more general nonlinear mapping between the given input-output signals, and to reduce the influence of outliers in measurements.

Feed forward neural networks (FFNNs) have found some applications to flight data analysis and aircraft modeling and control [3,4]. However many of such studies generally use back-propagation training algorithm which has slow convergence rate requiring many training sessions. Many applications require online decisions which should be based on processing of the data in real-time for which fast training algorithms must be used.

In this paper application of FFNNs to aerodynamic modeling/prediction and aircraft sensor failure detection using simulated data is investigated. For training of FFNNs, three algorithms: i) back-propagation (BPN), ii) fast back-propagation recursive least squares (RLS) filter with nonlinear output block (BPNRLSFLN) [5] and iii) fast back-propagator RLS filter with linear output block (BPNRLSFL) are implemented in PC MATLAB [6]. The results of aircraft sensor failure detection [7] using the fast BPNRLSFL scheme are also presented.

Feed Forward Neural Networks

FFNN is an information processing system consisting of a large number of simple processing elements. The elements are called artificial neurons or nodes. These neurons are interconnected by links which are characterized by the so called weights and cooperate to perform parallel distributed processing in order to solve a desired computational task. The name neural network comes from the fact that the background of the early researchers, who were involved in the study of functioning of the human brain and modeling of nervous systems, was in the field of biology, physiology and psychology rather than in computer science or engineering [2]. Eventhough ANNs have some resemblance to real neural networks, they are more appropriately called massively parallel adaptive filters/circuits (MAPAFs), as ANNs have technical roots in the area of analog computing/circuits and signal processing [2].

The input-output subspace modeling using the FFNNs is feasible because the basic (neural network) functions can adequately approximate the system behavior in an overall sense. FFNNs can be considered as nonlinear black-box model structures, the parameters (weights) of which can be estimated by conventional optimization methods. They are more suitable for system identification problems like: time-series modeling and prediction, sensor failure detection and estimation of aerodynamic coefficients.

* Scientist, Flight Mechanics and Control Division, National Aerospace Laboratories, Bangalore-560017, India

Manuscript received on 22 Aug 95; Paper reviewed and accepted on 22 Nov 95

* Paper to be presented at the 47th Annual General Meeting of the Aeronautical Society of India at IIT, Madras during 6-8 January 1996
Algorithms for Training **Feedforward Neural Networks**

An FFNNs have non-cyclic layered topology (Fig. 1) and can be considered to give structure-free (in the sense of conventional polynomial model) nonlinear mapping between the input-output signals of a given system. The FF " is first trained using the so called training set data. Then it is used for prediction using the same or different input set belonging to the same class of the data. This set is called the validation set. The weights of the network are estimated using the so called back-propagation gradient-based algorithm (BPN). Due to the layered structure, the estimation of weights of FFNN requires propagation of the error of the output layer in backward direction and hence the name BPN. This algorithm has slow convergence rate. The algorithm based on Kalman filter (BNRFLSF) is known to be faster than the BPN [5]. In this paper the algorithms are described using the matrix/vector notation for the sake of comparison, clarity and ease of implementation in PC MATLAB. Even if one does not have the NN toolbox of the PC MATLAB, these algorithm and the related simulation studies can be easily and efficiently carried out using the available (and newly formulated) dot-em (.m) files as is done for this paper. The variables of a typical FF " are described below:

\[
\begin{align*}
    u_o &= \text{input to the network (input layer)} \\
    n_i &= \text{the no. of input neurons (no. of inputs} u_o) \\
    n_h &= \text{the no. of neurons in the hidden layer} \\
    n_o &= \text{the no. of output neurons (no. of outputs} z) \\
    W_1 &= \text{weight matrix between input and hidden layer} \\
    W_{10} &= \text{bias weight vector} \\
    W_2 &= \text{weight matrix between hidden and output layer} \\
    W_{20} &= \text{bias weight vector} \\
    \mu &= \text{the learning rate parameter or step size}
\end{align*}
\]

**Back Propagation Algorithm**

Subscripts 0, 1 and 2 indicate input-, hidden- and output- layers respectively. During a forward pass training of the network, intermediate signals are computed for each layer and each node of the layer and weights of the network are estimated. The signal computation is done using the following equations (between input and hidden layer):

\[
\begin{align*}
    y_1 &= W_1 u_0 + W_{10} \\
    u_1 &= f(y_1)
\end{align*}
\]

Here \(u_1\) is the input to the hidden layer and \(f(y_1)\) is a nonlinear activation function. The signal computation between hidden and output layer is done using the following equations:

\[
\begin{align*}
    y_2 &= W_2 u_1 + W_{20} \\
    u_2 &= f(y_2)
\end{align*}
\]

Often, an unconstrained optimization problem for parameter estimation, is transformed into an equivalent system of differential equations which in turn constitute basic (neural network) algorithm to solve specific computational problems [6]:

\[
\frac{dW}{dt} = -\mu (t) \nabla E(W)
\]

where \(\nabla(.)\) denotes the gradient of the cost function \(E\) wrt \(W\) (the unknown parameters). With the output error defined as \(e_2 = z - u_2\), and a suitable quadratic cost function based on it, the evaluation of the gradient of \(E\) wrt \(W_2\) yields the following expression:

\[
\frac{\partial E}{\partial W_2} = -f(y_2) (z - u_2) u_1^T
\]

where \(u_1^T\) is the gradient of \(y_2\) wrt \(W_2\) [see Eq. (3)], and the derivative of "\(r\)" is given by:

\[
f(y) = \frac{2x e^{-\lambda y}}{(1 + e^{\lambda y})^2}
\]

The modified error for the output layer can be expressed as:

\[
e_{2b} = f'(y_2) (z - u_2)
\]

Combining Eqs. (5,6,8), the recursive weight-update rule for the output layer is obtained:

\[
W_{2}^{k+1} = W_{2}^{k} + \mu e_{2b}u_1^T + \Omega (W_{2}^{k} - W_{2}^{k-1})
\]
Here $\Omega$ is the momentum constant and the last term is used to smooth out the weight changes and to accelerate the convergence of the algorithm. Similarly the back-propagation of the error and the weight-update rule (for inner/hidden layer) are obtained as:

$$e_{1b} = f'(y_1) W_2^T e_{2b}$$

$$w_i^{1+1} = w_i^{1} + \mu e_{1b} + \Omega (w_i^{1} - w_i^{1-1})$$

Here $t$ is the data/sampling time index for recursive update of the estimates. The input-output data are presented to the network in sequential manner. This process is called pattern learning in the ANN terminology. Once all the data are scanned, they are presented again but with initial weights as the output from the previous session. The process is continued until the convergence is reached. One can call the entire process as the recursive-iterative method.

Back-propagation Recursive Least Squares Filter Algorithms

In these algorithms the weight update formulae for output layer and inner layers use the Kalman gains derived from the corresponding signals from the previous layers [5].

Algorithm with Nonlinear Output Layer (BPNRLSFNL)

Due to the presence of output nonlinearity, its inversion is required in order to employ the linear KF concepts to obtain this algorithm [5]. During the forward pass training of the network the signals $y$ and $u$ are computed for each layer as per the BPN algorithm using Eqs. (1-4). The Kalman gains, $K_{1,2}$ are computed for both the layers. The forgetting factors $h_{1,2}$ are chosen. The formulation is in the usual scalar data processing form.

For layer 1 the Kalman gain and covariance matrix update are given as [5]:

$$K_1 = P_1 u_0 (h_1 + u_0 P_1 u_0)^{-1}$$

$$P_1 = (P_1 - K_1 u_0 P_1) / h_1$$

For layer 2 the Kalman gain and covariance matrix update are given as:

$$K_2 = P_2 u_1 (h_2 + u_1 P_2 u_1)^{-1}$$

$$P_2 = (P_2 - K_2 u_1 P_2) / h_2$$

The modified output error is given as in the case of BPN algorithm by:

$$e_{2b} = f'(y_2) (z - u_2)$$

Back-propagation of the output error (for the inner/hidden layer) gives inner layer error:

$$e_{1b} = f'(y_1) W_2^T e_{2b}$$

Finally the weight-update rule (for output layer) is given as:

$$w_i^{1+1} = w_i^{1} + (d - y_2) K_2^T$$

where the components of $d$ are given as:

$$d_i = \frac{1}{\lambda} \ln(1 + z_i) (1 - z_i)$$

with $z_i \neq 1$. For the inner hidden layer, the weight update rule is given as:

$$w_i^{1+1} = w_i^{1} + \mu e_{1b} K_1$$

Except for the additional computation of Kalman gains for each layer, the procedure far training is the same as that for the BPN algorithm.

Algorithm with Linear Output Layer (BPNRLSFL)

In this algorithm the nonlinearities are used only for the inner layers and the output layer is kept as a linear block. The advantage of this modification is that it avoids the inversion of the output stage nonlinear function which might cause some ill-conditioning. In the present case, the use of linear KF concept is directly applicable. During the forward pass training of the network the signals $y$ and $u$ are computed for each layer as per the BPN algorithm using Eqs. (1-3). Since the output block is assumed to be linear, output signal is computed as follows:

$$u_2 = y_2$$

The Kalman gain related computations are done as per the CPNRNSFL algorithm. Since there is no output nonlinearity, the error for output layer is given by:

$$e_{2b} = e_2 = (z - y_2)$$
Back-propagation of the output error (to the inner/hidden layer) gives inner layer error:

$$e_{ib} = f(y_1)w_2^T e_{2b}$$  \hspace{1cm} (21)

The weight-update rules are given as:

for output layer: $$w_2^{t+1} = w_2^t + e_{2b}k_F$$  \hspace{1cm} (22)

for the inner/hidden layer: $$w_1^{t+1} = w_1^t + \mu e_{1b}k_F$$  \hspace{1cm} (23)

Once the data has been scanned and the convergence achieved, the estimated weights of the last iteration cycle are used and the input data are presented again to the network to obtain the predicted output. This output is compared with the desired/available output in order to judge the ability of the network for prediction. Although the mathematical description is given for FFNNs with one hidden layer, the algorithm are implemented for maximum of two hidden layers.

Application of the Feed forward Neural Network Algorithms

The simulation dam an: generated with sinusoidal signal input to the aircraft model [6]:

$$V = -\frac{\bar{q}}{m} C_D - g \sin(\theta - \alpha)$$

$$\dot{a} = -\frac{\bar{q}}{mV} C_L + \frac{\bar{q}}{V} \cos(\theta - \alpha) + q$$

$$\dot{q} = \frac{\bar{q} \times c}{I_y} C_m$$

$$\dot{\alpha} = q$$  \hspace{1cm} (24)

Here V is the aircraft velocity, $\alpha$ the angle of attack, $\theta$ the pitch attitude and $\bar{q}$ is the dynamic pressure. The constants $\bar{q}$, $C_D$, $m$ and $I_y$ are geometry and mass related parameters of the simulated aircraft. Aerodynamic Model is given as:

$$C_D = C_{D0} + C_{Da} \alpha + C_{D\bar{q}} \delta_e$$

$$C_L = C_{L0} + C_{La} \alpha + C_{L\bar{q}} \delta_e$$

$$C_m = C_{m0} + C_{ma} \alpha + C_{mc} \frac{\bar{q}}{V} + C_{m\bar{q}} \delta_e$$

The aircraft aerodynamics is represented for the purpose of modelling using FFNN as:

$$\dot{q} = h_2(V,\alpha,q,\delta_e)$$  \hspace{1cm} (27)

Here $h$ indicates some nonlinear functional relation. The signals $V, \alpha, q$ and $\delta_e$ are presented to the network as input patterns and signals $a$ and $q$ as output patterns. The FFNN is trained using the BPN and BPN-RLSF algorithms. In general BPN algorithm required more training sessions (10-15) than the fast BPN-RLSF algorithms.

Figure 2 shows the plots of time history match for prediction phase of FFNN using BPN algorithm for $\mu = 0.5, \Omega = 0.6$, ten hidden layer neurons (in a single hidden layer) and with scaled signals. Although the prediction is largely satisfactory, the algorithm is not able to predict the signals very accurately during the interval of 15 to 35 secs. Reasonably optimum values of training parameters and number of hidden layer neurons were used in all the cases. In Fig. 3 the results of the BPNRLSFL for the same data set are presented for $\mu = 40$. Six hidden layer neurons in a single hidden layer were used with scaled signals. The sigmoidal slope parameter $\lambda$ was chosen as 0.2. The performance of this algorithm is better than that of the BPN algorithm and only one training iteration was required. This is significant improvement over the classical gradient based BPN algorithm as far as the speed and accuracy are concerned.

![Fig 2 FFNN-BPN Algorithm Time History Match, Prediction Phase](image-url)
are used as the output and $a$ and $\delta_e$ as the input to the network. The results of this exercise using BPNRLSFL algorithm are shown in Fig. 4 (for $\mu = 0$ and six hidden layer neurons in a single hidden layer) with unscaled signals. Also the sigmoidal gain factors for the output layers were chosen different for each output : for $C_L$, $\lambda = 0.1$ and for $C_m$, $\lambda = 0.5$. For the inner layer the gain factor was chosen as 0.2. The $C_L$, plotted in Fig. 5 wrt angle of attack, tends to follow the pattern of the measured coefficient. Forgetting factors for RLS based algorithms were chosen as 0.99 for FFNN training in all the examples.

The main aim of this work is to illustrate the feasibility of using FFNNs for non-linear mapping of the aerodynamic coefficients (as a function of angle of attack, mach number, altitude, etc.) rather than estimating the aerodynamic derivatives. The approach should be useful in obtaining accurate non-linear aerodynamic models based on aerodynamic data (obtained from wind tunnels and analytical predictions) [4,8] in order to, perhaps, replace the time-consuming table look-up methods for nonlinear flight simulation exercises and related work. Accurate estimation of the aerodynamic derivatives using FFNNs requires: major modification in the cost function, training strategy and validation. This work has been recently initiated. Alternatively, for estimation of aerodynamic derivatives (like $C_L$, $C_m$, etc.) recurrent neural networks (FFNNs with some feed back from output to input of the network) can be considered [9].

Next, a scheme based on FFNN-BPNRLSFL algorithm is presented for detection of fault in aircraft sensors. Fig. 6 shows the scheme in comparison to one based on Kalman filter. The Kalman Filter/Observer based approach requires: i) specification of a model of the system, ii) integration of the system's dynamics, iii) computation of Kalman gain and covariances, iv) tuning of the filter (based on process/measurement covariance matrices), v) extended Kalman filter for nonlinear systems. However it is recursive and amenable to real-time online detection of faults. The FF ** based method has the following features: i) model of the system is not required, ii) hence the integration of the system equations is avoided, iii) computation of weights of the FF, iv) tuning of the non-linear-

Fig. 3 FFNN-BPNRLSFL Algorithm Time History Match, Prediction Phase

Fig. 5 Aerodynamic Coefficient (for a Segment of Time Scale) Prediction

Fig. 4 Aerodynamic Coefficient Modelling Using FFNN-BPNRLSFL Algorithm—Prediction Phase (with $a$ and $\delta_e$ as Input to FFNN)
predetermined based on the standard deviation of the residuals. Alternatively it can be taken as 2 to 3 times the steady value of the residual during the normal operating condition, after the FF has been sufficiently trained. This method is suitable for real-time online operation of the method.

Various phases of the NW operation for the sensor failure detection are shown in Fig. 7. The data used are the same as used in the exercises discussed above. The trapezoidal fault signal was added in the angle of attack time history. It can be seen from the figure that the NW is able to immediately detect the fault. For this study various parameters are: \( \mu = 40, \lambda = 0.2 \), forgetting factors \( = 0.99 \), hidden layers neurons \( = 6 \) in single hidden layer. The unscaled signals were used. The Fig. 7 shows all the phases of the NW operation during single session and iteration and it can be easily seen that the present scheme which combines the FFNN with BPNRLSF filter training algorithm is amenable to online detection of sensor failures. Fig. 8 shows the expanded view of the NW operation during the occurrence and detection phase (approx. 24-30 seconds interval) of the fault.

An estimate of the threshold is obtained in real-time as follows:

\[
v(k) = \frac{1}{k-1}(k-2)v(k-1) + r^2(5) \quad ; k = t_1, \ldots, t_1 + m
\]

\[\text{with } m = 40 \text{ to } 50\]  

\[\alpha_{th} = 3 \left[ \text{sqrt} (v(t_1 + m)) \right]\]

\[\text{Fig.6 Schematic Comparison of Sensor Fault Detection Schemes}\]

Here \( r \) is the residual generated by the FF. About 50 data points were required to obtain the steady value of the threshold during the normal operating condition, after the FF has been sufficiently trained. This method is suitable for real-time online operation of the method.

For fault-identification cross-correlation functions between various significant variables (like angular rates, etc.) can be evaluated and used [9]. Thus the scheme can be extended to detect faults in aircraft sensors as well as actuators. Any indication of fault thus determined will be useful to pilot to monitor the health of the aircraft flight control systems. Further validation is necessary for this
Concluding Remarks

In this paper feasibility of fast FFNN-BPNRLS filter schemes has been demonstrated for aerodynamic modeling/prediction and aircraft sensor failure detection. Quick online detection is feasible because of the use of fast BPNRLS filter which is used for training feedforward neural network to generate the residuals as validated by numerical simulation.

Acknowledgements

A part of the work was carried out by the author at the Institute of Flight Mechanics (IFM) under the DLR-CSIR collaborative exchange programme. The author is very grateful to Dr.-Ing. SM. P. Hamel, Director, DLR IFM, and Dr.-Ing. K.-F. Doherr, Head, Mathematical Methods and Data Handling Branch, DLR-IFM, for their support. Technical discussions with Dr. R.V. Jategaonkar, DLR-IFM, were very useful.


