Sādhanā, Vol. 25, Part 2, April 2000, pp. 193–203. © Printed in India

Parameter estimation using compensatory neural networks

M SINHA[1], P K KALRA[2] and K KUMAR[1]

[1]Department of Aerospace Engineering, and
[2]Department of Electrical Engineering, Indian Institute of Technology, Kanpur 208016, India
e-mail: kalra@iitk.ae.in

Abstract. Proposed here is a new neuron model, a basis for Compensatory Neural Network Architecture (CNNA), which not only reduces the total number of interconnections among neurons but also reduces the total computing time for training. The suggested model has properties of the basic neuron model as well as the higher neuron model (multiplicative aggregation function). It can adapt to standard neuron and higher order neuron, as well as a combination of the two. This approach is found to estimate the orbit with accuracy significantly better than Kalman Filter (KF) and Feedforward Multilayer Neural Network (FMNN) (also simply referred to as Artificial Neural Network, ANN) with lambda-gamma learning. The typical simulation runs also bring out the superiority of the proposed scheme over Kalman filter from the standpoint of computation time and the amount of data needed for the desired degree of estimated accuracy for the specific problem of orbit determination.

Keywords. Kalman filter; artificial neural networks; compensatory neural network architecture; higher order neuron; lambda-gamma learning; orbit determination.

1. Introduction

The neuron model comprising aggregation and thresholding functions is the basic unit of the Feedforward Multilayer Neural Network (FMNN). These neural networks have been successfully applied for prediction of time series problems (Werbos 1974; Lapedes & Farber 1987; Watrous & Shastri 1987; Sawai *et al* 1989; Jordan 1989; Kawato 1990; Narendra 1990; Nguyen & Widrow 1990). The basic approach for modeling through FMNN is based on determination of the appropriate architecture, i.e. number of hidden layers and number of neurons in each of the hidden layers. The number of hidden layers and the number of neurons in each layer determine the number of weights or the connection strength. The computational burden of a given optimization algorithm in turn depends on the number of weights to be computed. Various architectures that minimize the computational burden have been reported (Giles & Maxwell 1987). It is possible to reduce this burden by using compensatory neurons. It is felt that the proposed compensatory neuron, which is the basis for Compensatory Neural Network Architecture (CNNA), would not only reduce the

computational burden but also improve the quality of prediction. The example of an orbital parameter estimation problem has been considered to test the efficacy of this scheme and the result has been compared with the conventional neuron-based FMNN and Kalman filter (KF).

2. Pseudo-optimization strategy

The pseudo-optimization refers simply to a local iterative process with truncated Taylor series approximation to the function in the neighbourhood of the current point in the weight space. The steps involved in the algorithm proposed for pseudo-optimization that minimizes the error function are summarized below.

(i) Initialization of the network weights $\tilde{w}$ (generally random).
(ii) Determination of the search direction p_n and the step size α_n such that

$$E(\tilde{w}(n-1) + \alpha_n p_n) < E(\tilde{w}(n-1)). \tag{1}$$

(iii) Update of the weight vector using the relation

$$\tilde{w}(n) = \tilde{w}(n-1) + \alpha_n p_n. \tag{2}$$

(iv) Checking for $E'(\tilde{w}(n))$

(a) For $E'(\tilde{w}(n)) \neq 0$, n to be incremented by 1 and the step 2 to be repeated.
(b) If $E'(\tilde{w}(n)) = 0$, $\tilde{w}(n)$ to be taken as the required weight.

where

$$E(\tilde{w}(n)) = \sum_{k=1}^{K} 0.5(d_k - o_k)^2; \quad \text{error at the } n\text{th iteration} \tag{3}$$

$\tilde{w}(n) =$ weights in the connections at the nth iteration,
$d_k =$ desired output of the kth output node,
$o_k =$ actual output of the kth output node.

3. The lambda-gamma learning in feedforward multilayer neural network

In the algorithm presented in the last section, when the search direction is set to the negative gradient $-E'(\tilde{w})$ and the step size α_n taken as a constant (ε), we refer to it as the standard back-propagation algorithm. It is based on the well-known gradient descent algorithm. Here, the minimization is based on the first-order approximation, consequently, its convergence is slow. The neuronal nonlinearity used here is the sigmoidal function given by

$$y = 1/(1 + \exp^{-\lambda \text{net}}) \tag{4}$$

where

$$\text{net} = \sum w_{ji} y_i, \tag{5}$$

$\lambda =$ an arbitrarily chosen positive constant.

This type of training, however, involves considerable trial and error for fixing λ in order to ensure minimization of the error index. To overcome this problem, Zurada (1992)

proposed the lambda learning rule where λ was optimized during the learning phase. Later, Engelbrecht *et al* (1995) proposed lambda-gamma learning by introducing another parameter γ in the sigmoidal function, which now takes the form

$$y = \gamma/(1 + \exp^{-\lambda \text{net}}). \tag{6}$$

The new variable γ is also optimized during learning. This change is made to introduce "self-scaling" in the output units, thus doing away with the need for output normalization. Here, the permissible range, however, is limited to the interval $(-5, 5)$ for implementation. Later, Sinha *et al* (1998) used the tangent hyperbolic function given by

$$y = (2\gamma/(1 + \exp^{-\lambda \text{net}})) - \gamma, \tag{7}$$

and proposed suitable modifications in the lambda-gamma learning algorithm of Engelbrecht *et al* (1995). While applying it to the problem of the orbit determination, it was shown that the proposed modification makes it superior to the standard back-propagation algorithm. λ and γ are optimized during the learning phase, as before. The various pertinent relations involving the FMNN parameters are summarized in appendix A (Sinha *et al* 1998).

4. Proposed compensatory neural network architecture

The FMNN and the newly proposed CNNA are shown in figures 1–2. For the CNNA, separate neuron blocks have been taken. Each block has three layers of functions. In the first layer, in each of these blocks, the linear sum as well as that of all the weighted products of the input variables taken two at a time is carried out. In the second layer, only the summation is carried out. This sum is fed to the third layer where in each block two different neuronal nonlinearities and a summation function are present. Finally, the weighted sum from all the neurons is taken from the output nodes. The governing equations associated with the above architecture for implementation are presented below.

(i) *Output layer*

$$\Delta w_{kj}(n) = -\eta_1 \delta_{ok} y_j + \beta_1 \Delta w_{kj}(n-1), \tag{8}$$

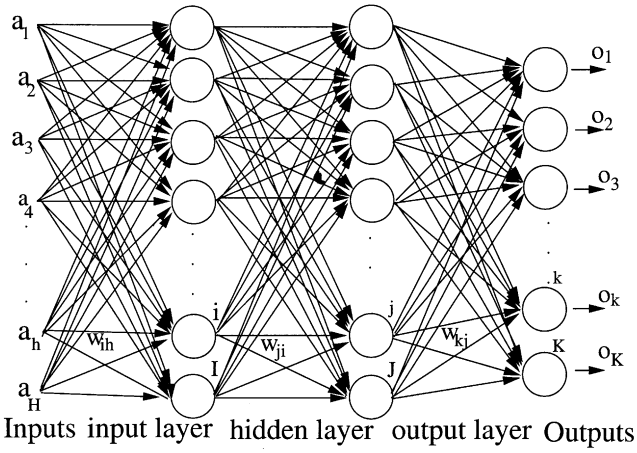

Figure 1. Schematic of standard feedforward neural network.

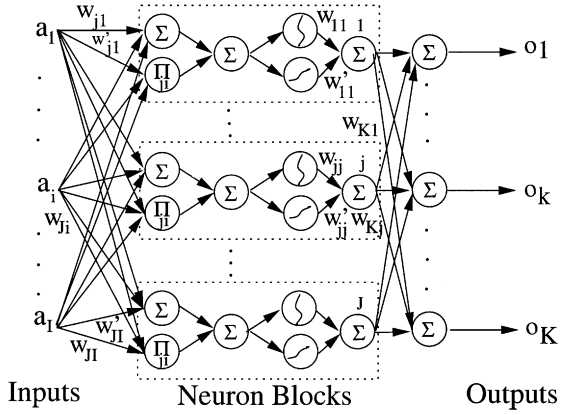

Figure 2. Schematic of feedforward compensatory neural network architecture (CNNA).

where

$$\delta_{ok} = -(d_k - o_k). \tag{9}$$

(ii) *Neuron block and input layer:*

$$\Delta w_{jj}(n) = -\eta_1(y_j - y'_j)\delta_j + \beta_1 \Delta w_{jj}(n-1), \tag{10}$$

$$\Delta \lambda_{y_j}(n) = -\eta_2 \text{net}_{y_j}\delta_{y_j} w_{jj}/\lambda_{y_j} + \beta_2 \Delta \lambda_{y_j}(n-1), \tag{11}$$

$$\Delta \gamma_{y_j}(n) = -\eta_3(y_j w_{jj}/\gamma_{y_j})\delta_j + \beta_3 \Delta \gamma_{y_j}(n-1), \tag{12}$$

$$\Delta w'_{jj}(n) = -\Delta w_{jj}(n), \tag{13}$$

$$\Delta \lambda_{y'_j}(n) = -\eta_2 \,\text{net}_{y'_j}\delta_{y'_j} w'_{jj}/\lambda_{y'_j} + \beta_2 \Delta \lambda_{y'_j}(n-1), \tag{14}$$

$$\Delta \gamma_{y'_j}(n) = -\eta_3(y'_j w'_{jj}/\gamma_{y'_j})\delta_j + \beta_3 \Delta \gamma_{y'_j}(n-1), \tag{15}$$

$$\Delta w_{ji}(n) = -\eta_1 \Omega_j a_i + \beta_1 \Delta w_{ji}(n-1), \tag{16}$$

$$\Delta w'_{ji}(n) = -\eta_1 \Omega_j a_i (sum_j - a_i w'_{ji}) + \beta_1 \Delta w'_{ji}(n-1), \tag{17}$$

where

$$w'_{jj} = 1.0 - w_{jj}, \tag{18}$$

$$\delta_j = \sum_{k=1}^{K} -(d_k - o_k) w_{kj}, \tag{19}$$

$$\delta_{y_j} = \lambda_{y_j}\delta_j(\gamma_{y_j}^2 - \gamma_j^2)/2\gamma_{y_j}, \tag{20}$$

$$\delta_{y'_j} = \lambda_{y'_j}\gamma_{y'_j}\delta_j/(1 + (\text{net}_{y'_j}\lambda_{y'_j})^2, \tag{21}$$

$$\Omega_j = \delta_{y_j} w_{jj} + \delta_{y'_j} w'_{jj}, \tag{22}$$

$$sum_j = \sum_{i=1}^{I} a_i w'_{ji}. \tag{23}$$

5. Example

In order to test the relative efficacy of the various neuron models presented earlier, the example of satellite orbit determination from observational data has been considered. For a

Table 1. Statistics of the observed data.

Observation variable	Standard deviation	Low bias	Medium bias	High bias
Range (m)	30	0	50	100
Range-rate (ms)	0.10	0.00	0.30	1.00
Azimuth (deg.)	0.03	0.00	0.03	0.01
Elevation (deg.)	0.03	0.00	0.03	0.01

more complete assessment, the estimates obtained using the well-known Kalman filter (see appendix B for algorithm) have also been included. The state vector considered here consists of three Cartesian components of the satellite position vector and three components of velocity vector and is given by

$$x = [x_1, x_2, x_3, x_4, x_5, x_6]^T,$$

where

$$x_1, x_2, x_3 = \text{Cartesian components of position vector,}$$

$$x_4, x_5, x_6 = \text{velocity components.}$$

It may be added that the state vector at any time instant uniquely defines the six orbital parameters (called orbital elements) that completely determine the orbit. Similarly, given the six orbital parameters/elements one can determine the state vector at any point of time. The observational vector is taken as

$$y = [\rho, \dot{\rho}\,\mathrm{Az}, \mathrm{El}]^T.$$

6. Results and discussion

Observational data are generated, using the inverse square gravitational field model of the earth at regular intervals of one second. The data points duly incorporate the random errors

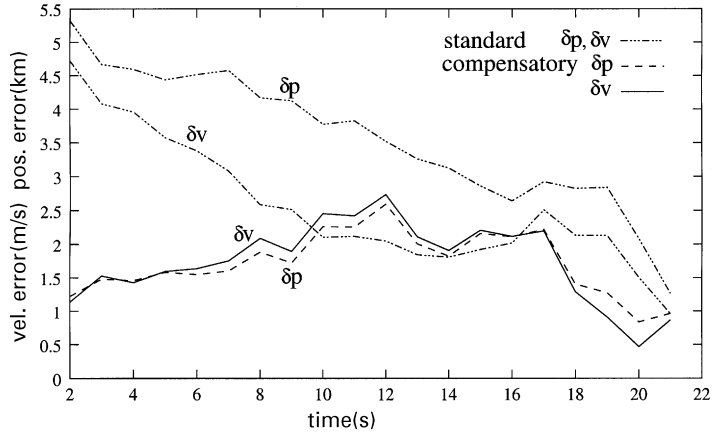

Figure 3. Plots showing propagation of position (δp) and velocity (δv) errors for ANN trained without bias and predicting zero bias case.

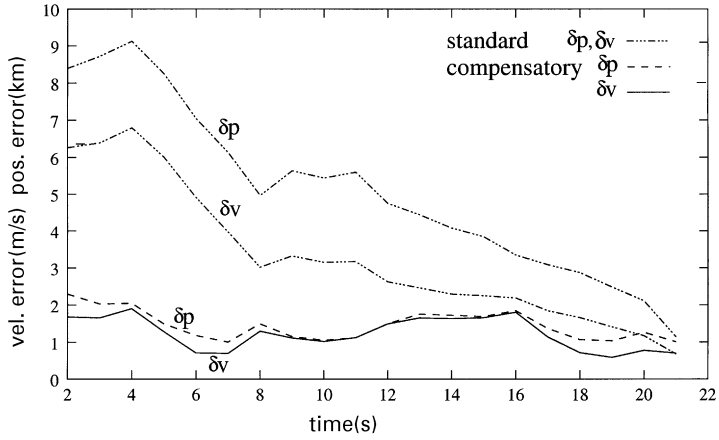

Figure 4. Plots showing propagation of position (δp) and velocity (δv) errors for ANN trained with high bias and predicting zero bias case.

as well as biases in the measurements of observations. Three different levels of biases are assumed as shown in table 1.

The biases are treated as unknown and assumed unavailable for processing of the data. The initial position error assumed is ~ 500 km, while the initial velocity error is taken in the range ~ 300 m/s. After some trials, the number of neurons in the FMNN is fixed as 12 in the input layer, 14 in the hidden layer and 1 in the output layer. The CNNA has 6 blocks. Each block consists of neurons, with tangent hyperbolic and arctangent, respectively. Thus the total number of connections in the FMNN model is 353 and the number of neurons 27, while in the CNNA the total number of connections is 174 and of neurons 12. After some trial and error, the values of learning and momentum rates are chosen and the number of iterations for training both, the FMNN as well as CNNA, set at 1500. The results of the backpropagation developed have been presented in figures 3–8. It is interesting to note that, even though the eventual levels of precision achieved are pretty close in the two situations,

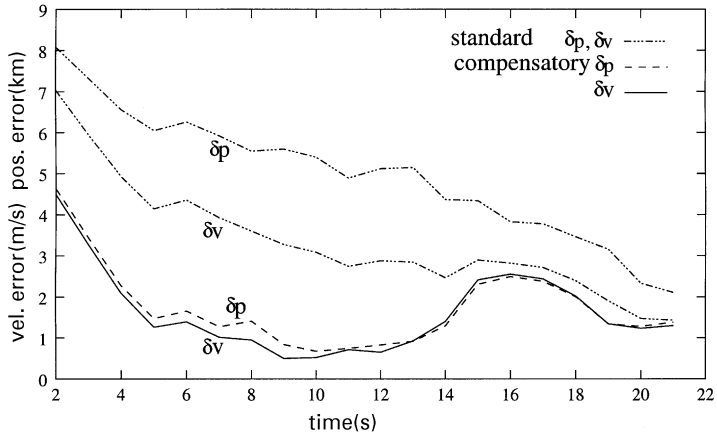

Figure 5. Plots showing propagation of position (δp) and velocity (δv) errors for ANN trained without bias and predicting medium bias case.

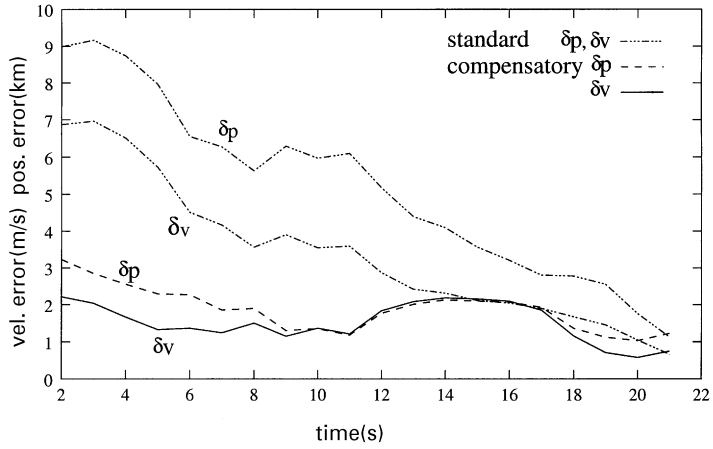

Figure 6. Plots showing propagation of position (δp) and velocity (δv) errors for ANN trained with high bias and predicting medium bias case.

CNNA enables higher precision much earlier than FMNN. Besides, CNNA has the added advantage of significantly fewer connections. Here, the number of neurons is just half that for the FMNN architecture. Table 2 presents the errors in the estimated orbital elements from observational data using the Kalman filter and CNNA.

In general, KF is found to require much larger data size for attaining the desired order of precision in estimating elements. In this paper, Kalman filter results presented are based on 120 data sampling points spanning over 120 s. In contrast, for neural network simulations, much smaller data size spanning over, say, the first 20 seconds was found to be adequate and hence used for implementation. Low, medium and high biases were used for generating the observation data used for estimation. It may be observed that the estimated orbital parameter accuracies achieved through CNNA are in general significantly higher than those obtained using the Kalman filter. Even with substantially reduced number of neurons, as indicated earlier, CNNA takes about 50% of the computational time needed for the

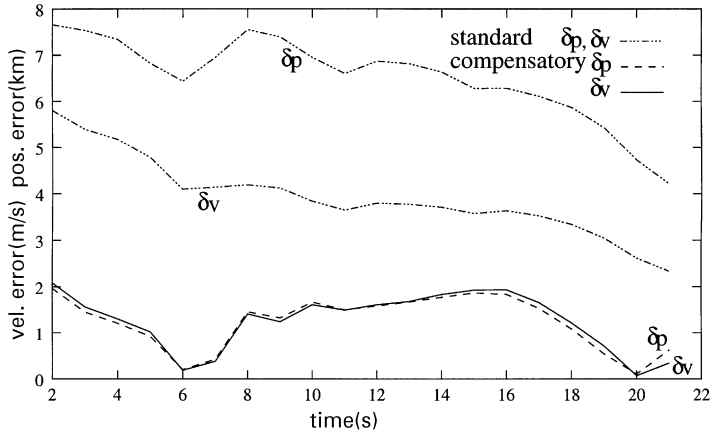

Figure 7. Plots showing propagation of position (δp) and velocity (δv) errors for ANN trained without bias and predicting high bias case.

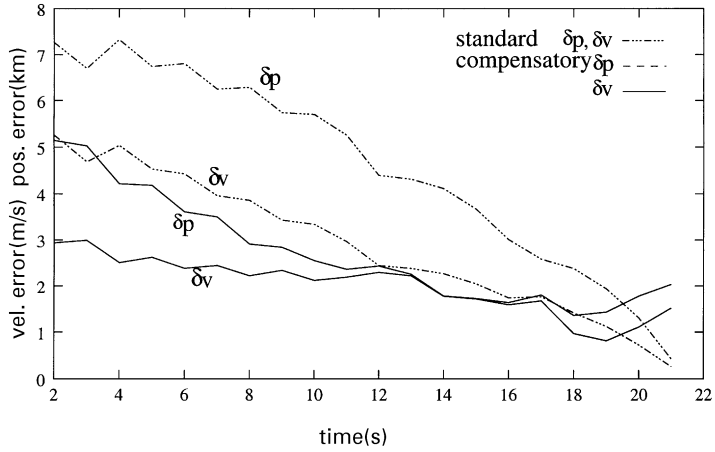

Figure 8. Plots showing propagation of position (δp) and velocity (δv) errors for ANN trained with high bias and predicting high bias case.

FMNN architecture considered. During the training of neural networks, optimal weights are obtained. These are now utilized for testing of the prediction dataset. The consequent reduced processing time during the prediction phase leads to significant computational time-saving as compared to that for the Kalman filter. Besides, CNNA leads to significantly higher prediction accuracies.

7. Concluding remarks

This paper presents a novel approach incorporating the higher neuron model in the FMNN. The proposed architecture (CNNA) combined with the lambda–gamma learning scheme makes the first-order learning rather attractive. It clearly outperforms the FMNN architecture, especially as regards the computational efficiency. It is felt that combining this architecture with the second-order optimization scheme for the backpropagation of errors, may significantly reduce the computational time of second-order learning schemes and their memory requirements.

Table 2. Errors in different orbital parameters.

	Bias level	Δr_a (km)	Δr_p (km)	$\Delta i \times 10^4$ (rad)	$\Delta\omega \times 10^4$ (rad)	$\Delta\Omega \times 10^4$ (rad)	$\Delta\theta \times 10^4$ (rad)
KF	Zero	0.22	5.38	1.8268	28.2212	0.3756	−28.5674
CNNA*	Zero	1.70	−0.18	−0.0647	−0.1804	−0.0920	0.1341
CNNA**	Zero	−1.17	−0.35	−0.3992	0.0808	−0.5678	0.3664
KF	Medium	−1.46	7.89	2.3669	47.5490	−1.9016	−46.6830
CNNA*	Medium	1.07	0.49	0.2385	−0.0336	0.3393	−1.6689
CNNA**	Medium	−1.42	−0.39	−0.4178	0.1236	−0.5943	0.0257
KF	High	−4.55	9.22	6.5212	38.0850	−6.0325	−35.5513
CNNA*	High	0.34	−0.28	0.0521	0.4824	0.0741	0.2460
CNNA**	High	−2.34	−0.70	−0.8438	0.2704	−1.2004	0.9555

*Training without any bias; **training with arbitrarily chosen high bias

Appendix A. FMNN with lambda-gamma learning

A1. *Output layer*

$$\Delta w_{kj}(n) = -\eta_1 \delta_{o_k} y_j + \beta_1 \Delta w_{kj}(n-1), \tag{A1}$$

$$\Delta \lambda_{o_k}(n) = -\eta_2 \,\mathrm{net}_{o_k} \delta_{o_k} / \lambda_{o_k} + \beta_2 \Delta \lambda_{o_k}(n-1) \tag{A2}$$

$$\Delta \gamma_{o_k}(n) = \eta_3 (d_k - o_k) o_k / \gamma_{o_k} + \beta_3 \Delta \gamma_{o_k}(n-1), \tag{A3}$$

where

$$\delta_{o_k} = -(d_k - o_k)(\gamma_{o_k}^2 - o_k^2)\lambda_{o_k} / 2\gamma_{o_k}. \tag{A4}$$

A2. *Hidden layer*

$$\Delta w_{ji}(n) = -\eta_1 x_i \delta_{y_j} + \beta_1 \Delta w_{ji}(n-1), \tag{A5}$$

$$\Delta \lambda_{y_j}(n) = -\eta_2 \,\mathrm{net}_{y_j} \,\delta_{y_j} / \lambda_{y_j} + \beta_2 \Delta \lambda_{y_j}(n-1), \tag{A6}$$

$$\Delta \gamma_{y_j}(n) = -\eta_3 y_j \left(\sum_{k=1}^{K} \delta_{o_k} w_{kj} \right) \Bigg/ \gamma_{y_j} + \beta_3 \Delta \gamma_{y_j}(n-1), \tag{A7}$$

where

$$\delta_{y_j} = \lambda_{y_j} ((\gamma_{y_j}^2 - y_j^2)/2\gamma_{y_j}) \sum_{k=1}^{K} \delta_{o_k} w_{kj}. \tag{A8}$$

A3. *Input layer*

$$\Delta w_{ih}(n) = -\eta_1 a_h \delta_{x_i} + \beta_1 \Delta w_{ih}(n-1), \tag{A9}$$

$$\Delta \lambda_{x_i}(n) = -\eta_2 \,\mathrm{net}_{x_i} \delta_{x_i} / \lambda_{x_i} + \beta_2 \Delta \lambda_{x_i}(n-1), \tag{A10}$$

$$\Delta \gamma_{x_i}(n) = -\eta_3 (x_i / \gamma_{x_i}) \sum_{j=1}^{J} \delta_{y_j} w_{ji} + \beta_3 \Delta \gamma_{x_i}(n-1), \tag{A11}$$

where

$$\delta_{x_i} = \lambda_{x_i} ((\gamma_{x_i}^2 - x_j^2)/2\gamma_{x_i}) \sum_{j=1}^{J} \delta_{y_j} v_{ji}. \tag{A12}$$

Appendix B. Extended Kalman filter with adaptive driving noise covariance matrix

B1. *State equation*

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f[x(t), t] + G(t) w(t). \tag{B1}$$

B2. *Measurement equation*

$$y = h[x(t), (t)] + \nu(t). \tag{B2}$$

B3. *Time update*

$$\hat{x}_{k+1/k} = \hat{x}_{k/k} + \int_{t_k}^{t_{k+1}} f[x_{k/k}, t]\mathrm{d}t, \tag{B3}$$

$$P_{k+1/k} = F_k P_{k/k} F_k^T + G_k Q_k G_k^T. \tag{B4}$$

B4. *Measurement update*

$$K_{k+1} = P_{k+1/k} H_{k+1}^T [H_{k+1} P_{k+1/k} H_{k+1}^T + R_{k+1}]^{-1}, \tag{B5}$$

$$P_{k+1/k+1} = [I - K_{k+1} H_{k+1}] P_{k+1/k}, \tag{B6}$$

$$\hat{x}_{k+1/k+1} = \hat{x}_{k+1/k} + K_{k+1}[y_{k+1} - h(\hat{x}_{k+1/k}, t_k)]. \tag{B7}$$

B5. *Adaptive driving noise model*

An adaptive driving noise model based on the state residuals used is described below (Kumar *et al* 1988).

$$\begin{aligned} Q_{ij} &= 0; \quad i \neq j, \\ &= |\Delta x_i \cdot \Delta \dot{x}_i| \Delta t; \quad i = j. \end{aligned} \tag{B8}$$

List of symbols

Az	azimuth,
a_h	input,
d_k	desired output,
El	elevation,
F_k	state transition matrix at time t_k, = Jacobian $[\partial f/\partial x]_k$,
G_k	state noise gain matrix at time t_k,
H_k, H_{k+1}	observation matrix at time t_k, t_{k+1} = observation matrix $[\partial h/\partial x]_k$,
Δ_i	error in angle of inclination,
K_k, K_{k+1}	Kalman gain matrix at time t_k, t_{k+1},
k	kth interval of time,
o_k	output of output layer neuron,
$P_{k/k}$	state covariance matrix at time t_k, given measurements to t_k,
$P_{k+1/k}$	state covariance matrix at time t_{k+1}, given measurements to t_k,
Q_k	driving noise covariance matrix at time t_k,
R_k, R_{k+1}	observation noise covariance at time t_k, t_{k+1},
Δr_a	error in apogee distance,
Δr_p	error in perigee distance,
t	time,
Δt	step size on time,
Δw	weight correction,
x_i	output of the input layer,
$\hat{x}_{k/k}$	state estimate at time t_k, given measurements to t_k,
$\hat{x}_{k+1/k}$	state estimate at time t_{k+1}, given measurements to t_k,
Δx_i	computed state residual,
$\Delta \dot{x}_i$	time derivative of state residual computed numerically through curve fitting,

y	neuron-output,
y_j	output of hidden layer neuron,
β_1	momentum factor for weights,
β_2	momentum factor for steepness factor,
β_3	momentum factor for multiplication factor,
γ	multiplication factor,
$\Delta\gamma$	multiplication factor correction during nth iteration,
η_1	learning rate for weights,
η_2	learning rate for steepness factor,
η_3	learning rate for multiplication factor,
$\Delta\theta$	error in time of perigee passage,
λ	steepness factor,
$\Delta\lambda$	steepness factor correction during nth iteration,
ρ	range,
$\dot{\rho}$	range-rate,
$\Delta\Omega$	error in nodal angle,
$\Delta\omega$	error in argument of perigee.

References

Engelbrecht A P, Cloete I, Geldenhuys J, Zurada J M 1995 Automatic scaling using gamma learning for feedforward neural networks. From natural to artificial computing. *Lecture Notes Comput. Sci.* 930: 374–381

Giles C L, Maxwell T 1987 Learning, invariance, and generalization in high order networks. *Appl. Opt.* 26: 4972–4978

Jordan M 1989 Generic constraints on under specified target trajectories. In *Proc. IEEE Int. Joint Conf. Neural Networks*

Kawato M 1990 Computational schemes and neural network models for formation and control of multijoint arm trajectory. In *Neural networks for robotics and control* (eds) W T Miller, R Sutton, P Werbos (Cambridge, MA: Mass. Inst. Technol. Press)

Kumar K, Yadav D, Srinivas B V 1988 Some efficient noise models for extended Kalman filter. In *39*th *Congress of the International Astronautical Federation*, Bangalore, 1–5

Lapedes A, Farber R 1987 Nonlinear signal processing using neural networks prediction and system modelling, Los Alamos National Laboratory, Los Alamos, LA-UR-262

Narendra R 1990 Adaptive control using neural networks, In *Neural networks for robotics and control* (eds) W T Miller, R Sutton, P Werbos (Cambridge, MA: MIT Press)

Nguyen D, Widrow B 1990 The truck backer-upper: An example of self-learning in neural networks. In *Neural networks for robotics and control* (eds) W T Miller, R Sutton, P Werbos (Cambridge, MA: MIT Press)

Sawai H, Waibel A, Hafiner P, Miyatake M, Shikano K 1989 Parallelism, hierarchy, scaling in time delay neural networks for spotting Japanese phonemes/CV-syllables. In *Proc. IEEE Int. Joint Conf., Neural Networks*

Sinha M, Kumar K, Kalra P K 1998 Neural network alternative to Kalman filter for satellite orbit determination. *IEEE Aerosp. Electron. Syst.* (submitted)

Watrous R, Shastri L 1987 Learning phonetic features using connectionist networks: an experiment in speech recognition. In *Proc. 1st IEEE Int. Conf., Neural Networks*

Werbos P 1974 *Beyond regression: New tools for prediction and analysis in the behavioural science*. PhD dissertation, Committee on Appl. Math., Harvard Univ., Cambridge, MA

Zurada J M 1992 Lambda learning rule for feedforward neural networks. *Proc. IEEE Int. Conf., Neural Networks*, San Fransisco, California