Water Resources Management **18:** 143–161, 2004.

River Flow Forecasting using Recurrent Neural Networks

D. NAGESH KUMAR[1]*, K. SRINIVASA RAJU[2] and T. SATHISH[3]
[1] *Department of Civil Engineering, Indian Institute of Science, Bangalore, India;* [2] *Civil Engineering Department, Birla Institute of Technology and Science, Pilani, India;* [3] *Department of Civil Engineering, Indian Institute of Technology, Kharagpur, India*
(* *author for correspondence, e-mail: nagesh@civil.iisc.ernet.in)*

(Received: 2 April 2003; in final form: 29 December 2003)

Abstract. Forecasting a hydrologic time series has been one of the most complicated tasks owing to the wide range of data, the uncertainties in the parameters influencing the time series and also due to the non availability of adequate data. Recently, Artificial Neural Networks (ANNs) have become quite popular in time series forecasting in various fields. This paper demonstrates the use of ANNs to forecast monthly river flows. Two different networks, namely the feed forward network and the recurrent neural network, have been chosen. The feed forward network is trained using the conventional back propagation algorithm with many improvements and the recurrent neural network is trained using the method of ordered partial derivatives. The selection of architecture and the training procedure for both the networks are presented. The selected ANN models were used to train and forecast the monthly flows of a river in India, with a catchment area of 5189 km^2 up to the gauging site. The trained networks are used for both single step ahead and multiple step ahead forecasting. A comparative study of both networks indicates that the recurrent neural networks performed better than the feed forward networks. In addition, the size of the architecture and the training time required were less for the recurrent neural networks. The recurrent neural network gave better results for both single step ahead and multiple step ahead forecasting. Hence recurrent neural networks are recommended as a tool for river flow forecasting.

Key words: forecasting, hydrologic time series, recurrent neural networks, river flows

1. Introduction

The effectiveness of any decision depends upon the nature of a sequence of events preceding the decision. The ability to predict the uncontrollable aspects of these events prior to making the decision should permit an improved choice over that which can otherwise be made. The purpose of forecasting is to reduce the risk in decision making. Information regarding stream flow, at any given point of interest, is necessary in the analysis and design of several water resources projects such as dam construction, reservoir operation, flood control and wastewater disposal. The most widely used stochastic models for river flow forecasting belong to the class of ARIMA (Auto Regressive Integrated Moving Average) models proposed by Box and Jenkins (1976) (e.g. Mujumdar and Nagesh Kumar, 1990).

Preliminary concepts of artificial neural networks (ANNs) and their adaptability to hydrology are very well explained in ASCE (2000a) and Govindaraju and Rao (2000). An exhaustive list of references on ANN applications in hydrology is given in ASCE (2000b). Kang *et al*. (1993) used ANNs and autoregressive moving average models to predict daily and hourly streamflows in a river basin in Korea. Karunanithi *et al*. (1994) estimated streamflows at an ungauged site based on data from stream gauging stations located 30 km upstream and 20 km downstream of the sampling site. Markus *et al*. (1995) used ANNs with the back-propagation algorithm to predict monthly streamflows at a gauging station in Southern Colorado, with the inputs as snow water equivalent alone, or snow water equivalent and temperature. Raman and Sunilkumar (1995) modelled multivariate monthly hydrologic time series using ANNs and compared the results with those obtained from a statistical model. Thirumalaiah and Deo (1998) used ANNs to forecast river stage. Recurrent neural networks (RNNs) were used in Hydrology and related fields. Gong *et al*. (1996) used RNNs for solid transport modelling in sewer systems during storm events. Chow *et al*. (1997) developed a recurrent sigma-P neural network model for rainfall forecasting in Hong Kong. Anamala *et al*. (2000) found that RNNs performed better when compared with other architectures of ANNs for predicting watershed runoff.

Even though the knowledge of how an ANN actually works is not fully available, preliminary results have been quite encouraging. They also offer the following advantages:

- The application of a neural network does not require a priori knowledge of the underlying process.
- All the existing complex relationships between various aspects of the process under investigation need not be known.
- ANNs are data driven when compared to conventional approaches, which are model driven.

1.1. WORKING OF AN ARTIFICIAL NEURAL NETWORK

There are basically three different layers present in most ANNs (Vemuri, 1988). The first layer contains input neurons which receive input from the external world. The layer next to the input layer is the hidden layer, and this layer may consist of one or more layers of neurons with the higher layers receiving input from the immediately preceding layers. The final layer of the network is the output layer. The neurons present in this layer give the output of the network. Artificial neurons (ANs) receive their inputs from a number of other ANs or from outside world. A weighted sum of these inputs constitutes the argument of an activation function. This function is assumed to be nonlinear. Hard limiting i.e., either the step or the signum function threshold and soft limiting i.e., the sigmoidal, are the most often used forms of non linearities. The resulting value of the activation function is the

output of the AN. This output is distributed along weighted connections to other ANs. The notion of memory in a conventional computer is analogous to the concept of weight settings. As these weighted connections play an important role, ANNs are also called connectionist models of computation.

1.2. ARTIFICIAL NEURAL NETWORKS MODELS

ANN models are specified by the net topology, node characteristics, training and learning rules. These rules specify an initial set of weights and indicate how weights should be adapted to improve performance. Both design procedures and the training algorithms are the topics of much current research. The Hopfield model was the first model proposed in the field of ANNs and is used for pattern recognition. In the single layer perceptron (Rosenblatt, 1959), the input layers are directly connected to the output layer. Multi layered perceptrons (Rosenblatt, 1959) are feed forward nets with one or more layers between the input and output nodes. The additional layers contain hidden units or nodes that are not directly connected to the input layers or output layers. Multi layer perceptrons overcame many of the limitations of single layered perceptrons, but were not generally used in the past because of the non availability of training algorithms. The capabilities of multi layered perceptrons stem from the nonlinearities within the nodes. It was observed that as the number of layers (or nodes in the hidden layer) between the input and output nodes increases, more complex problems could be solved using the network. The cascade correlation algorithm is an efficient constructive training algorithm developed by Fahlman and Lebiere (1990). This algorithm combines the idea of incremental architecture and learning in its training procedure. In brief, training starts with a minimal network consisting of an input and output layer. The dynamic expansion of the network continues until the problem is successfully learned. Thus the cascade correlation algorithm automatically constructs a suitable architecture for a given problem (Karunanithi *et al.*, 1994). Recurrent neural networks are of recent origin in ANN models.

2. Backpropagation Algorithm

The Backpropagation algorithm (Rumelhart *et al.*, 1986) is the most popular of the ANN training algorithms. The Backpropagation algorithm (BPA) is basically a procedure to train feed forward models (FFMs). The FFMs are those type of models in which the outputs can be sent only to the immediate next layers. A typical feed forward network is shown in Figure 1. The process of selecting a suitable architecture for a required problem can be broadly classified into three steps (Carling, 1995): 1. Fixing the architecture, 2. Training the network and 3. Testing the network. All these steps and the procedure for training the network are well documented in literature (ASCE, 2000a). Many improvements are suggested in the

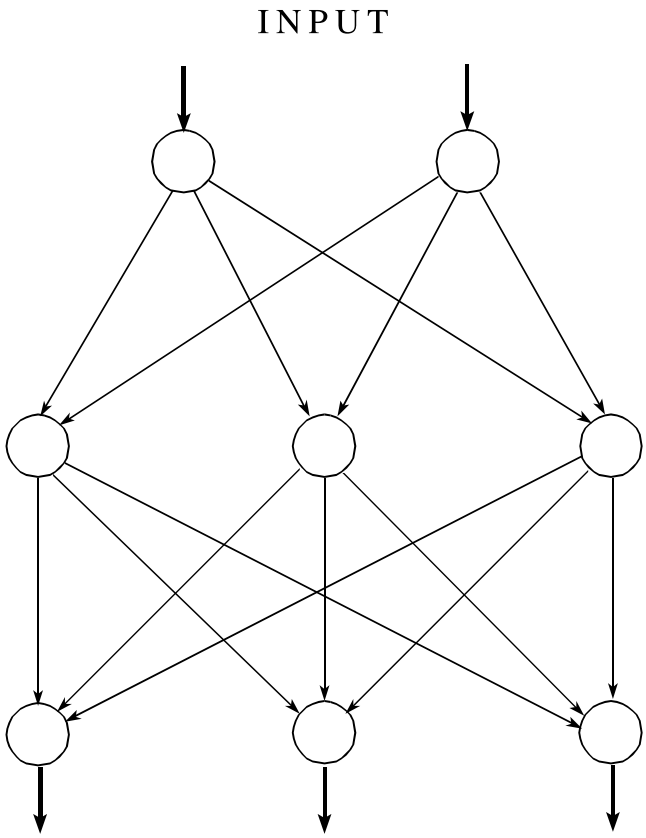

Figure 1. Typical Feed Forward Neural Network.

literature for the conventional Backpropagation algorithm and some are discussed in the next section and they are implemented for the present study.

2.1. IMPROVING BPA TRAINING USING OPTIMUM LEARNING RATE AND MOMENTUM

This method aims to improve the BPA algorithm by changing the learning rate and momentum terms (Yu and Chen, 1997). Once the error for each pattern is determined, the procedure is to find new values for the learning rate and momentum rate after each iteration. The optimum values are found by differentiating the output

values for each pattern with respect to the learning rate and the momentum.

$$\mu_{opt} = \frac{\sum_{p=1}^{N_p}\left(\frac{\partial fout_p}{\partial \mu}\right)^t * \Delta_p * \sum_{p=1}^{N_p}\left|\frac{\partial fout_p}{\partial \beta}\right|^2 - \sum_{p=1}^{N_p}\left(\frac{\partial fout_p}{\partial \beta}\right)^t * \Delta_p * \sum_{p=1}^{N_p}\left(\frac{\partial fout_p}{\partial \mu}\right)^t * \left(\frac{\partial fout_p}{\partial \beta}\right)}{\sum_{p=1}^{N_p}\left|\frac{\partial fout_p}{\partial \mu}\right|^2 * \sum_{p=1}^{N_p}\left|\frac{\partial fout_p}{\partial \beta}\right|^2 - \sum_{p=1}^{N_p}\left[\left(\frac{\partial fout_p}{\partial \mu}\right)^t * \left(\frac{\partial fout_p}{\partial \beta}\right)\right]^2} \tag{1}$$

$$\beta_{opt} = \frac{\sum_{p=1}^{N_p}\left(\frac{\partial fout_p}{\partial \beta}\right)^t * \Delta_p * \sum_{p=1}^{N_p}\left|\frac{\partial fout_p}{\partial \mu}\right|^2 - \sum_{p=1}^{N_p}\left(\frac{\partial fout_p}{\partial \mu}\right)^t * \Delta_p * \sum_{p=1}^{N_p}\left(\frac{\partial fout_p}{\partial \beta}\right)^t * \left(\frac{\partial fout_p}{\partial \mu}\right)}{\sum_{p=1}^{N_p}\left|\frac{\partial fout_p}{\partial \mu}\right|^2 * \sum_{p=1}^{N_p}\left|\frac{\partial fout_p}{\partial \beta}\right|^2 - \sum_{p=1}^{N_p}\left[\left(\frac{\partial fout_p}{\partial \mu}\right)^t * \left(\frac{\partial fout_p}{\partial \beta}\right)\right]^2} \tag{2}$$

where

Δ_p = $(aout_p - fout_p)$ taking $\mu = 0$ and $\beta = 0$;
$aout_p$ = the actual output expected from the network for the pattern p;
$fout_p$ = the computed final output from the network for the pattern p;
N_p = the number of patterns;
μ = the learning rate generally in the range of 0 to 1;
β = the momentum.

2.2. IMPROVING BPA USING EXTENDED BACKPROPAGATION

In conventional backpropagation, the output from a neuron is obtained by using a sigmoidal function. The output is generally of the form $1/(1 + \exp^{-x*\text{net}})$. The value of x is generally taken to be unity. In this method, the BPA is improved by performing a gradient descent on the steepness parameter, x as well (Sperduti and Starita, 1993). The new value of x is obtained by differentiating the system error, E, with respect to the steepness parameter.

$$x_{\text{new}} = x_{\text{old}} - \varepsilon * \left(\frac{-\partial E}{\partial x_{\text{old}}}\right) , \tag{3}$$

where ε is a small positive constant. A value of 1.5 is used for ε in this article.

$$\frac{\partial E}{\partial x_{\text{old}}} = \frac{\partial E}{\partial fout} * \frac{\partial fout}{\partial x_{\text{old}}} \tag{4}$$

$$\frac{\partial fout}{\partial x_{\text{old}}} = \left(\frac{\exp^{x_{\text{old}} * inp}}{(1 + \exp^{x_{\text{old}} * inp)}}\right)^2 * inp \tag{5}$$

$$E = \frac{1}{N_p}\sum_{k=1}^{N_p}\left(aout_k - fout_k\right)^2 . \tag{6}$$

Where, system error, E, for all the patterns (N_p) is calculated as the sum of the squared error for each pattern.

2.3. IMPROVING BPA THROUGH DYNAMIC SELF ADAPTATION

The dynamic self adaptation algorithm is a two step procedure. The first step induces a mutation on the learning rate and compares the new value with that corresponding to the previous situation and then selects the best (Salomon and Hemmen, 1996). Steepest descent algorithms can be used without normalisation or with normalisation. The weight change with normalisation for the iteration k, is done with the help of the following equation.

$$w\,(k+1) = w\,(k) - \frac{\nabla E(w)}{\nabla\,|E(w)|} = w(k) - \eta_{k+1} * e_k\ , \tag{7}$$

where e_k is the unit vector in the direction of $\nabla E(w)$. The new learning rate is

$$\begin{aligned} \eta_{(k+1)} &= \eta_k \zeta \quad \text{if } E(w_k - \eta_k e_k \zeta) \leq E(w_k - \eta_k e_k/\zeta) \\ &= \eta_k/\zeta \quad \text{otherwise} \end{aligned} \quad . \tag{8}$$

Out of the two possible outcomes from the algorithm, the one that gives lower value for the system error is used. The value of ζ is recommended as 1.8 for an optimum solution (Salomon and Hemmen, 1996).

In the present work, the conventional BPA algorithm has been implemented with all the improvements explained above.

3. Recurrent Neural Networks

Forecasting of hydrologic time series is based on the previous values of the series depending on the number of persistence components (memory). Recurrent neural networks (RNN) provide this facility through number of feed back loops. A generalised RNN can send input in either direction from and to all the layers. Thus the output of the network not only depends on the external inputs it receives but also on the state of the network in the previous time step.

There are essentially three ways that 'memory' can be introduced into static neural networks (ASCE, 2000a). These are (in increasing order of complexity and capability):

1. Tapped delay line models: The network has past inputs explicitly available to determine its response at a given point in time (Mozer, 1989).
2. Context models or partial recurrent models: These models retain the past output of nodes instead of retaining the past raw inputs (Kothari and Agyepong, 1997) and

3. Fully recurrent models: These models employ full feedback and interconnections between all nodes (Almeida, 1988).

Context models are used in the present study. Islam and Kothari (2000) provide a brief overview of recurrent backpropagation including its mathematical aspects and implementation details.

3.1. ADVANTAGES OF RNNs OVER FFNs

The RNNs possess important features, namely, events from the past can be retained and used in current computations. They also allow the network to produce complex, time varying outputs in response to simple static inputs. It has also been observed that RNNs rarely settle in local minima even though no precautions are taken to this effect (Carling, 1995).

3.2. TRAINING METHODOLOGY IN RNNs

The method of training RNNs is similar to that of the FFN models. The training algorithm is explained with the help of a simple example. A small network, which has two input neurons, one hidden layer having three neurons and one output neuron, is shown in Figure 2. In addition to this, a neuron taking input from the output layer and connected to the hidden layer is added as shown. This neuron is the additional neuron in RNNs.

Initially the weights assigned are small random numbers. The inputs are presented to the input units and the output from the network is calculated just as in the case of the FFN. The process is repeated for all the patterns. After finding the system error, the training of the network is based on the steepest gradient method using the method of ordered partial derivatives. Method of ordered partial derivatives as given by Piche (1994) is explained briefly in the next section.

3.3. METHOD OF ORDERED PARTIAL DERIVATIVES

To define the ordered partial derivatives, the concept of ordered set of equations should be known. Let $[Z_1, Z_2, \ldots, Z_n]$ be a set of n variables whose values are determined by a set of n equations. This set of equations is defined as an ordered set of equations if each variable Z_i is a function of the variables $[Z_1, Z_2, \ldots, Z_{i-1}]$. Because of the ordered nature of this set of equations, the variables $[Z_1, Z_2, \ldots, Z_{i-1}]$ must be calculated before Z_i can be calculated. As an example the following three equations form a set of ordered equations:

$$Z_1 = 1 \tag{9}$$

$$Z_2 = 3Z_1 \tag{10}$$

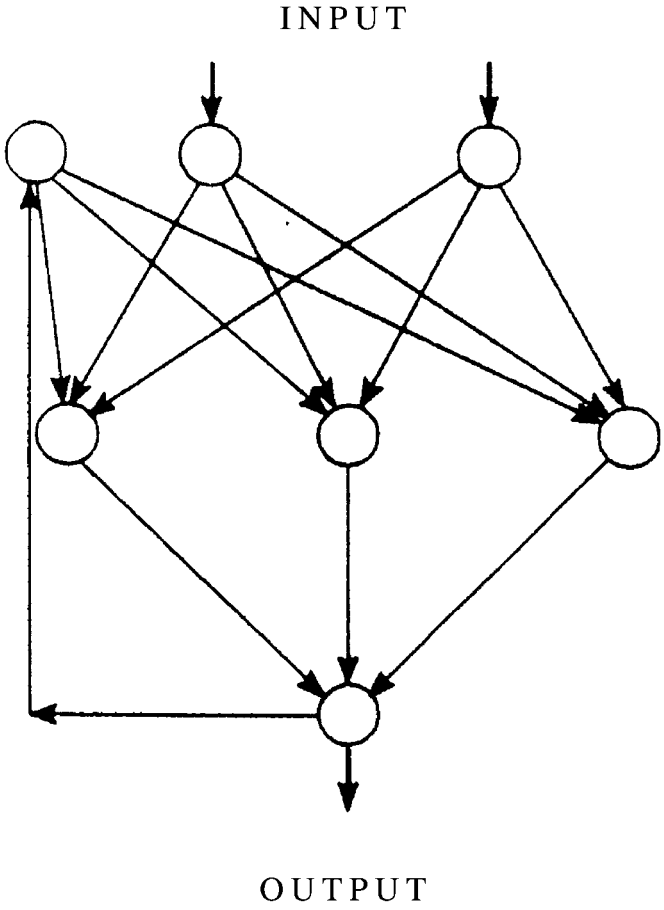

Figure 2. Typical Recurrent Neural Network.

$$Z_3 = Z_1 + 2Z_2 \,. \tag{11}$$

When calculating a partial derivative it is necessary to specify which of the variables are held constant and which are allowed to vary. If this is not specified it is assumed that all the variables are held constant except those terms appearing in the denominator of the partial derivative. Thus the partial derivative of Z_3 with respect to Z_1 is $\partial Z_3/\partial Z_1$. An ordered partial derivative (OPD) is a partial derivative whose constant and varying terms are determined using an ordered set of equations. The constant terms of the OPD of Z_j with respect to Z_i (denoted as $\partial^+ Z_j/\partial Z_i$ to distinguish from the ordinary partial derivative) are $[Z_1, Z_2, \ldots, Z_{i-1}]$ and the varying terms are $[Z_i, \ldots, Z_j, \ldots, Z_n]$.

Thus

$$\begin{aligned}
&\text{If } j \le i \qquad \frac{\partial^+ Z_j}{\partial Z_i} = 0 \\
&\text{If } j = i + 1 \quad \frac{\partial^+ Z_j}{\partial Z_i} = \frac{\partial Z_j}{\partial Z_i} \\
&\text{If } j > i + 1 \quad \frac{\partial^+ Z_j}{\partial Z_i} = \frac{\partial Z_j}{\partial Z_i} + \sum_{k=i+1}^{j-1} \left(\frac{\partial^+ Z_j}{\partial Z_k} * \frac{\partial Z_k}{\partial Z_i} \right) \\
&\text{or} \qquad\qquad \frac{\partial^+ Z_j}{\partial Z_i} = \frac{\partial Z_j}{\partial Z_i} + \sum_{k=i+1}^{j-1} \left(\frac{\partial Z_j}{\partial Z_k} * \frac{\partial^+ Z_k}{\partial Z_i} \right)
\end{aligned} \tag{12}$$

For the particular example chosen

$$\frac{\partial^+ Z_3}{\partial Z_1} = \frac{\partial Z_3}{\partial Z_1} + \frac{\partial Z_3}{\partial Z_2} * \frac{\partial Z_2}{\partial Z_1} = 1 + 2*3 = 7 \,. \tag{13}$$

For training the network, once the system error has been calculated, the weights are changed according to the following equation.

$$w(k+1) = w(k) - \mu * \frac{\partial^+ E}{\partial w(k)} \,, \tag{14}$$

where

$w(k)$	=	the weight at the present iteration, k;
$w(k+1)$	=	the corresponding weight to be used for the next iteration, $k+1$;
$\frac{\partial^+ E}{\partial w(k)}$	=	the gradient of the system error with respect to the weight $w(k)$.

The procedure for calculating the gradient of the system error is as follows.

$$\frac{\partial^+ E}{\partial w(k)} = \frac{\partial E}{\partial w(k)} + \sum_{k=1}^{N_p} \left(\frac{\partial E}{\partial fout_k} * \frac{\partial^+ fout_k}{\partial w(k)} + \frac{\partial E}{\partial w(k)} * \frac{\partial^+ w(k)}{\partial w(k)} \right) \,, \tag{15}$$

where the term $\partial E / \partial w(k)$ is equal to 0 because E is not a direct function of the weights. Therefore the equation reduces to

$$\frac{\partial^+ E}{\partial w(k)} = \sum_{k=1}^{N_p} \left(\frac{\partial E}{\partial fout_k} * \frac{\partial^+ fout_k}{\partial w(k)} \right) \,, \tag{16}$$

where

$$\frac{\partial E}{\partial fout_k} = (-2)*(aout_k - fout_k)$$

$$\frac{\partial^+ fout_k}{\partial w(k)} = \frac{\partial fout_k}{\partial w(k)} + \sum_{j=0}^{k-1}\left(\frac{\partial fout_k}{\partial W_j(k)} * \frac{\partial^+ W_j(k)}{\partial w(k)}\right) + \sum_{j=0}^{k-1}\left(\frac{\partial fout_k}{\partial fout_j} * \frac{\partial^+ fout_j}{\partial w(k)}\right) . \quad (17)$$

The terms in this equation are computed as follows.

The second term of the equation is calculated using the chain rule expansion. Here W indicates the weight vector. The term $\partial fout_k/\partial W_j(k)$ of the first summation is non zero only when $k = j$; therefore this summation contains only one non-zero term. As a result, the first summation can be written as $\partial fout_k/\partial w(k)$. In addition the first term of the second summation, $\partial fout_k/\partial fout_j$ is non zero only when $(k - j) < L$, where L is the number of previous output values which are sent as inputs to the hidden layer. Using these results Equation (17) can be simplified as

$$\frac{\partial^+ fout_k}{\partial w(k)} = \frac{\partial fout_k}{\partial w(k)} + \sum_{j=1}^{L}\left(\frac{\partial fout_k}{\partial fout_{k-j}} * \frac{\partial^+ fout_{k-j}}{\partial w(k)}\right) . \quad (18)$$

This equation is used recursively to calculate the output gradient for all the patterns. It is initialised using

$$\frac{\partial^+ fout_1}{\partial w(k)} = \ldots = \frac{\partial^+ fout_L}{\partial W(k)} = 0 . \quad (19)$$

The gradient of the error for a neuron j in the hidden layer and the neuron k in the output layer is calculated as follows.

$$\frac{\partial fout_k}{\partial w2_{j,k}} = \left(\frac{\exp^{inp2_k}}{(1+\exp^{inp2_k})^2}\right) * out1_j , \quad (20)$$

where

$inp2_k$ = the total input received by the neuron k for the particular pattern.

$out1_j$ = the output of the neuron j for that pattern.

and

$$\frac{\partial fout_k}{\partial fout_j} = \left(\frac{\exp^{inp2_k}}{(1+\exp^{inp2_k})^2}\right) * wr_j , \quad (21)$$

where wr_j is the weight which transfers the output of pattern j to the hidden layer. Once the values of the above two equations are known the weights can be changed accordingly.

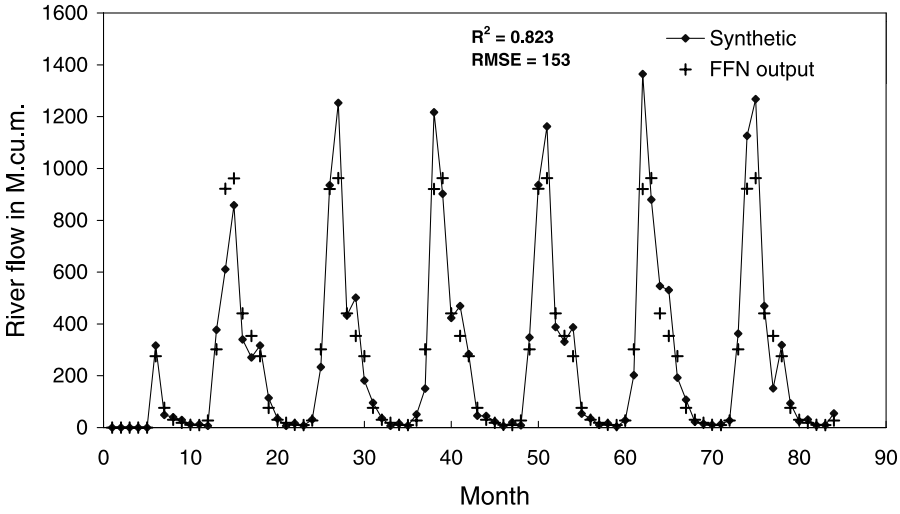

Figure 3. Comparison of FFN forecast with synthetic data.

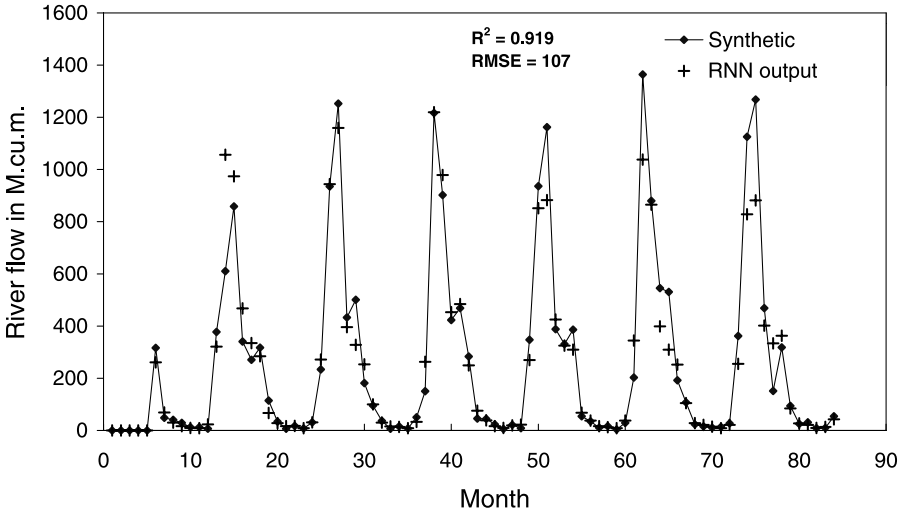

Figure 4. Comparison of RNN forecast with synthetic data.

The gradient of the weights between the input layer neuron i and the hidden layer neuron j is calculated as follows.

$$\frac{\partial fout_k}{\partial w1_{i,j}} = \left(\frac{\exp^{inp2_k}}{(1+\exp^{inp2_k})^2}\right) * w2_{j,k} * \left(\frac{\exp^{inp}\, 1_j}{(1+\exp^{inp1_j})^2}\right) * inp_{k,i} \ . \tag{22}$$

Once the gradient of the error for all weights is calculated, the weights are changed and the process repeated until the errors converge to a pre-specified value or for a specified maximum number of iterations.

Table I. Monthly mean and standard deviation of Hemavathi River flows of 57 years

Month	Mean (M.cu.m)	Standard Deviation
June	195.39	178.24
July	650.96	488.03
August	710.71	424.47
September	322.85	144.87
October	234.67	184.57
November	203.85	106.63
December	58.23	30.12
January	25.19	8.65
February	13.70	9.50
March	8.13	4.18
April	8.01	5.52
May	18.73	17.37

4. Model Application

4.1. TESTING WITH SYNTHETIC DATA

Initially FFNs and RNNs are tested on synthetically generated data using an ARMA model. For the Hemavathi River monthly flow series, an ARMA (1,2) was identified as the best model based on maximum likelihood criterion by Mujumdar and Nagesh Kumar (1990). This model and the corresponding parameters were used for synthetic generation. The use of synthetic data will enable the performance of the ANN models to be tested without noise or observational errors that may be present in the actual data. This testing is done for a seven year period (84 months). Figure 3 shows the comparison between FFN predictions and the synthetic data. RNN predictions are compared with synthetic data in Figure 4. Two statistics are used to compare the forecasts obtained from ANN with actual data viz., root mean square error (RMSE) and R^2 values. These statistics obtained both for FFN and RNN predications are shown in Figures 3 and 4, respectively. It can be noticed from these two figures that the RNNs are able to preserve the variation in monthly data much better than the FFNs.

4.2. CASE STUDY

The FFN and RNN models have been applied to forecast the monthly flows of the river Hemavathi, Karnataka state, India. Hemavathi is a tributary of the river Cauvery in peninsular India. The gauging site is located at Akkihebbal in Karnataka state. The catchment area up to this gauging site is 5189 km^2. Monthly flow data at

Table II. Network configuration for the FFN and RNN models

Variable	FFN	RNN
Number of input units	5	5
Number of hidden layers	2	2
Number of units in first hidden layer	14	10
Number of units in second hidden layer	14	10
Number of output units	1	1
Number of previous output values passed to first hidden layer	–	3
Value of learning rate	0.01	0.90
Value of the momentum rate	0.2	0.00
Number of iterations performed for training	50000	50000
Root mean squared value of pattern error	0.161	0.154

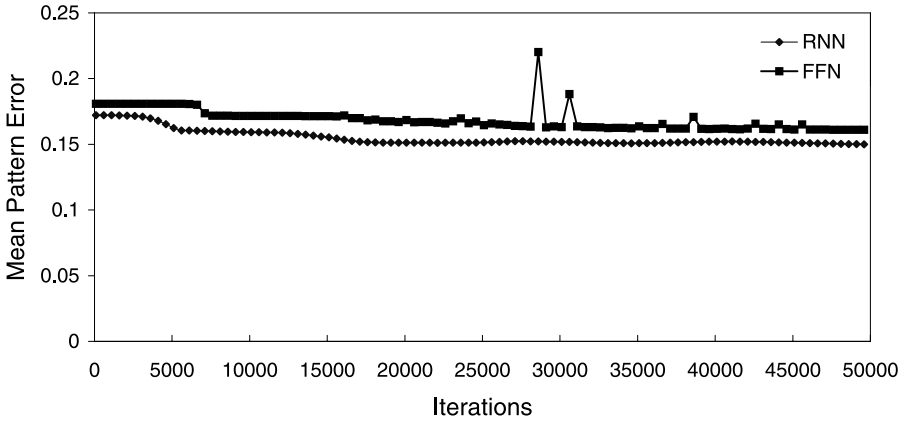

Figure 5. Variation of mean pattern error for RNN and FFN with number of iterations.

the site from 1916–1917 to 1972–1973, i.e. for 57 yr, is available. The monthly flow ranges from 1.84 million cubic metres (M.cu.m.) in summer months to 2894.81 M.cu.m. in monsoon months. Monthly mean and standard deviation values obtained from the 57 yr data are given in Table I. The first 50 yr data are used for training and the remaining 7 yr data are used for testing.

4.3. STANDARDIZATION OF THE DATA

The monthly data are used for training the network after standardization (subtracting monthly mean and dividing it by standard deviation of the corresponding month) to remove the cyclicity or periodicity present in the data. As the activation function is a sigmoidal function, the data are scaled between 0 and 1. The resulting data are then used for training.

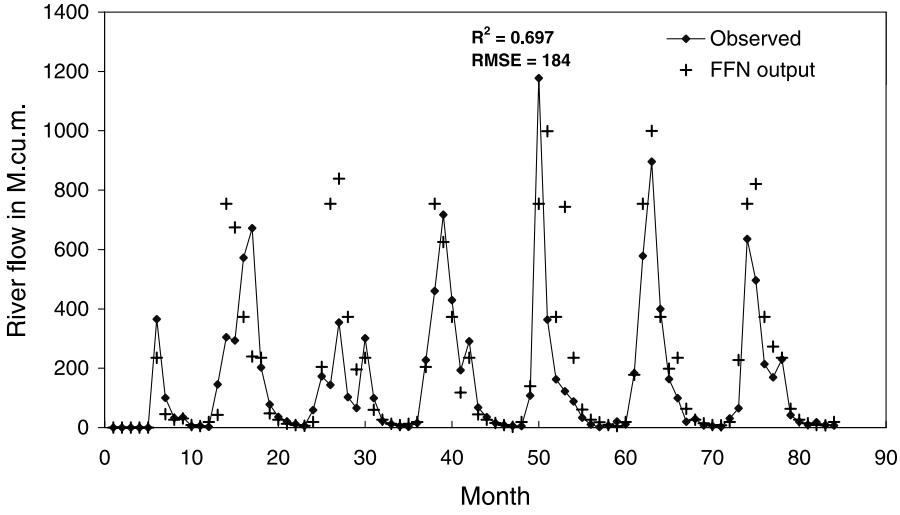

Figure 6. Comparison of FFN output with observed data of the training set.

4.4. FIXING THE ANN ARCHITECTURE

Initially, the architecture had three layers viz., one input layer, one hidden layer and one output layer. For Hemavathy monthly river flows, Mujumdar and Nagesh Kumar (1990) have observed that there are significant autocorrelations upto 3 lags and low autocorrelations from lag 3 to 5 implying that there is a strong memory of 3 months and week memory of two months. Therefore number of neurons in the input layer was varied from 3 to 5 for the FFN (tapped delay line models). Similarly for RNN, number of previous output values passed to first hidden layer varied from 3 to 5. The number of hidden layer neurons was varied from 5 to 20. The number of neurons in the output layer was fixed as 1. It was observed that this architecture with a hidden layer of 20 neurons was not sufficient for training. The architecture was then modified to 4 layers introducing an additional hidden layer. Details of the final ANN architectures used for FFN and RNN are given in Table II. The network was trained for a number of iterations, ranging from 20,000 to 1,00,000. It was observed that after 50,000 iterations the mean pattern error was fluctuating instead of further converging. Therefore, for training, the upper limit for the number of iterations was fixed at 50,000. Figure 5 shows the variation of the mean pattern error for the FFN and RNN models with respect to the number of iterations.

4.5. TRAINING OF THE NETWORK

Details of the FFN and RNN configurations are given in Table II. As already mentioned, the first 50 yr data were used for training. The inputs were presented to the input nodes and the final values were obtained after training. The output values

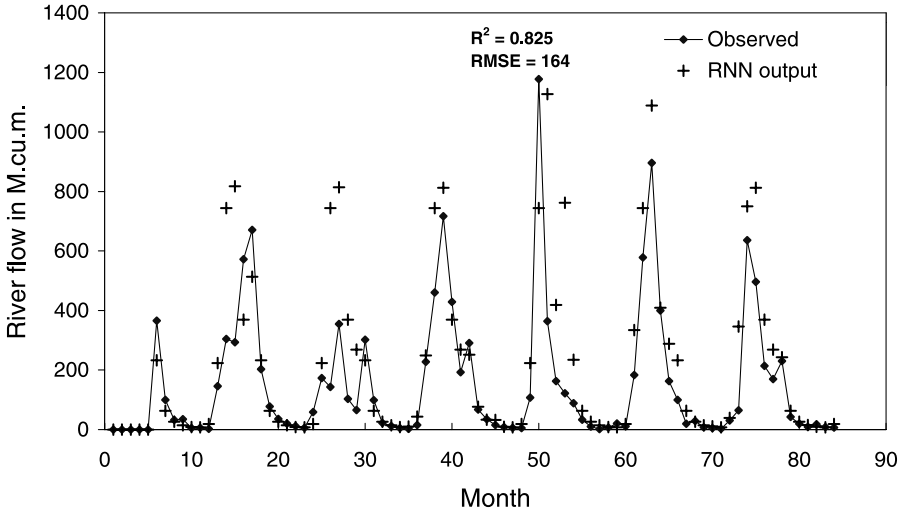

Figure 7. Comparison of RNN output with observed data of the training set.

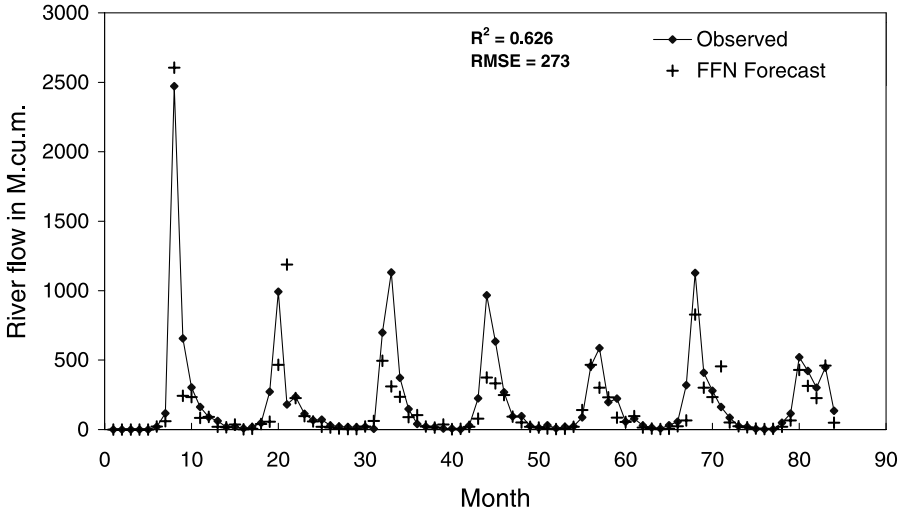

Figure 8. Comparison of one step ahead forecast using FFN with observed data.

from the network were then back transformed by reversing the procedure explained earlier to get the monthly flow data. Figures 6 and 7 show the actual flow data and the predicted flow data obtained by the FFN and the RNN models, respectively, after completion of training. For the sake of clarity, instead of showing the training results for the entire 50 yr, results are shown only for a 7 yr period within the 50 yr period in Figures 6 and 7 along with respective R^2 and RMSE values. It can be noticed from these two figures that the RNN has performed better than the FFN.

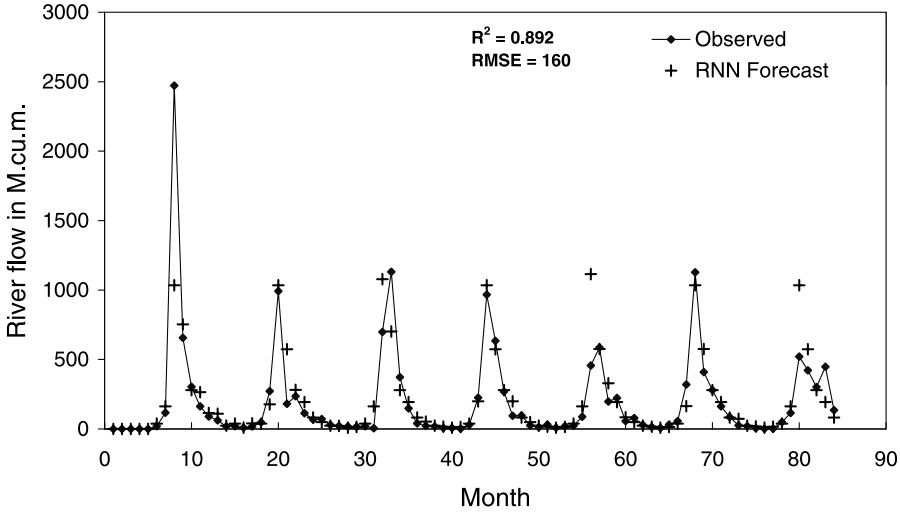

Figure 9. Comparison of one step ahead forecast using RNN with observed data.

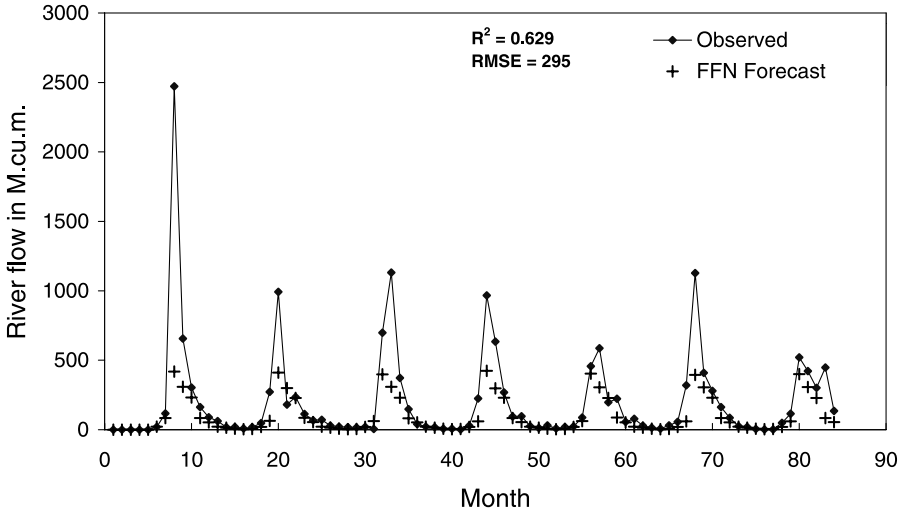

Figure 10. Comparison of multiple-step ahead forecast using FFN with observed data.

4.6. TESTING OF THE NETWORK

The final values of weights, obtained after training, are used as the connection weights for testing the networks. Testing is done for a period of 7 yr. Both the single step ahead forecasting (i.e. forecasting data only for one month at a time) and multiple step ahead forecasting are done. For a single step ahead forecasting, the inputs are updated after every month and the network is re-trained. For multiple step ahead forecasting, the network is updated after the end of every year. The performance of the network is assessed based on comparison between the actual values and the output values of the network. Figures 8 and 9 show the results

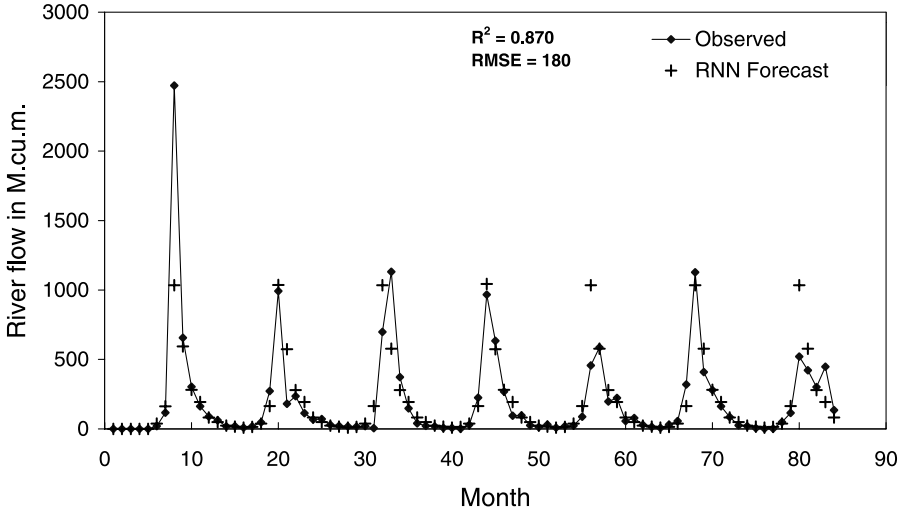

Figure 11. Comparison of multiple-step ahead forecast using RNN with observed data.

obtained by the FFN and RNN models when the networks were used for one step ahead forecasting along with concurrent observed flows. Corresponding values of R^2 and RMSE are also shown in the figures. In one step ahead forecasting network weights are updated after every time step considering the observed value for that period. In multiple-step ahead forecasting, predictions were made for 12 time steps (1 year) and at the end of the year, network weights are updated considering the observed values for that year. Figures 10 and 11 show the results obtained when the models were used for multiple-step ahead forecasting along with observed flows. Corresponding values of R^2 and RMSE are also shown in the figures. From these quantitative statistics it can be seen that the RNN performed better when compared to the FFN both in one step ahead forecasting and multiple step ahead forecasting.

5. Conclusions

Neural networks offer several advantages over conventional approaches. The most important aspect is their ability to develop a generalised solution to a problem from a given set of examples and to combine and adapt to changing circumstances with exposure to new variations in the problem. This attribute of generalisation permits them to be applied to a variety of problems and to produce valid solutions even when there are errors in training data or in the description of the problem. In this study, monthly streamflows of the river Hemavathi, India, were forecast using two types of ANN viz., Feed Forward Neural Networks and Recurrent Neural Networks. The RNNs were found to perform better for forecasting monthly river flows. Environmental impact of hydrology on human activities demands monitoring for reducing impacts of extreme events such as floods and droughts which in

turn require development of real time forecasting models. It is envisaged that this work will contribute to this end.

Acknowledgements

This research work is supported by Department of Science and Technology, Govt. of India under the project number HR/OY/E-13/95 sanctioned to the first author.

References

Almeida, L. B.: 1988, 'Backpropagation in Perceptrons with Feedback', R. Eckmiller and Ch. Von der Malsburg (eds), *Neural Computers*, Springer Verlag, Berlin, pp. 199–208.

Anmala, J., Zhang, B. and Govindaraju, R. S.: 2000, 'Comparison of ANNs and empirical approaches for predicting watershed runoff', *J. Water Resour. Plann. Manage., ASCE* **126**(3), 156–166.

ASCE Task Committee on Application of Artificial Neural Networks in Hydrology: 2000a, 'Artificial neural networks in hydrology. I: Preliminary concepts', *J. Hydrol. Engineer. ASCE* **5**(2), 115–123.

ASCE Task Committee on Application of Artificial Neural Networks in Hydrology: 2000b, 'Artificial neural networks in hydrology. II: Hydrologic applications', *J. Hydrol. Engineer., ASCE* **5**(2), 124–137.

Bishop, C.M.: 1995, *Neural Networks for Pattern Recognition*, Oxford University Press, New York.

Box, G. E. P. and Jenkins, G. M.: 1976, *Time Series Analysis: Forecasting and Control*, Holden-Day Publications, San Francisco, U.S.A.

Carling, Alison: 1995, *Introduction to Neural Networks*, Galgotia Publications, New Delhi.

Chow, T. W. S. and Cho, S. Y.: 1997, 'Development of a recurrent sigma-P neural network rainfall forecasting system in Hong Kong', *Neural Comput. Appl.* **5**(2), 66–75.

Fahlman, S. E. and Lebiere, C.: 1990, 'The Cascaded-Correlation Learning Architecture', *Rep. CMU-CS-90-100*, Carnegie Melon University, Pittsburgh.

Gong, N., Denoeux, T. and Bertrand Krajewski, J. L.: 1996, 'Neural networks for solid transport modeling in sewer systems during storm events', *Water Sci. Technol.* **33**(9), 85–92.

Govindaraju, R. S. and Rao, A. R. (eds): 2000, *Artificial Neural Networks in Hydrology*, Kluwer Academic Publishers, Amsterdam.

Islam, S. and Kothari, R.: 2000, 'Artificial neural networks in remote sensing of hydrologic processes', *J. Hydrol. Engineer., ASCE* **5**(2), 138–144.

Kang, K. W., Kim, J. H., Park, C. Y. and Ham, K. J.: 1993, 'Evaluation of Hydrological Forecasting System Based on Neural Network Model', *Proc., 25th Congress of Int. Assoc. for Hydr. Res.*, International Association for Hydraulic Research, Delft, The Netherlands, pp. 257–264.

Karunanithi, N., Grenney, W. J., Whitley, D. and Bovee K.: 1994, 'Neural networks for river flow forecasting', *J. Comput. Civil Engineer., ASCE* **8**(2), 201–218.

Kothari, R. and Agyepong, K.: 1997, 'Induced Specialization of Context Units for Temporal Pattern Recognition and Reproduction', J. Principe, L. Gile, N. Morgan and E. Wilson (eds), *Proc., IEEE Neural Networks for signal Processing VII*, Institute of Electrical and Electronics Engineers, New York, pp. 131–140.

Magoulas, G. D.: 1997, 'Effective back propagation training with variable step size', *Neural Networks* **10**(1), 69–82.

Markus, M., Salas, J. D. and Shin, H.-K.: 1995, 'Predicting Streamflows Based on Neural Networks', *Proc., 1st Int. Conf. on Water Resour. Engrg.*, ASCE, New York, pp. 1641–1646.

Mozer, M. C. and Smolensky, P.: 1989, 'Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment', in D. Touretzky (ed.), *Advances in Neural Information Processing Systems 1*, Morgan Kaufmann, San Monteo, California, pp. 107–115.

Mujumdar, P. P. and Nagesh Kumar, D.: 1990, 'Stochastic models of streamflow – Some case studies', *Hydrol. Sci. J.* **35**, 395–410.

Piche, S. W.: 1994, 'Steepest descent algorithms for neural net controllers and filters', *IEEE Transactions on Neural Networks* **5**(2), 198–212.

Raman, H., and Sunilkumar, N.: 1995, 'Multi-variate modeling of water resources time series using artificial neural networks', *Hydrol. Sci. J.* **40**, 145–163.

Rumelhart, D. E, Hinton, G. E. and Williams, R. J.: 1986, 'Learning internal representation by back-propagating errors', *Nature* **323**, 533–536.

Rosenblatt, R.: 1959, *Principles of Neuro Dynamics*, Spartan Books, New York.

Salomon, R. and Hemmen, J. L. V.: 1996, 'Accelerating back propagation through dynamic self-adaptation', *Neural Networks* **9**(4), 589–601.

Sperduti, A. and Starita, A.: 1993, 'Speed up learning and network optimization with extended back propagation', *Neural Networks* **6**, 365–383.

Thirumalaiah, K. and Deo, M. C.: 1998, 'River stage forecasting using artificial neural networks', *J. Hydrol. Engineer., ASCE* **3**(1), 26–32.

Vemuri, V.: 1988, 'Artificial Neural Networks: An Introduction', in *Artificial Neural Networks: Theoretical Concepts*, The Computer Society of the IEEE, pp. 1–12.

Yu, X.-H. and Chen, G.-A.: 1997, 'Efficient back propagation learning using optimal learning rate and momentum', *Neural Networks* **10**(3), 517–527.