DETERMINING NUMBER OF INPUT NODES OF RECURRENT NEURAL NETWORKS FOR RIVER FLOW FORECASTING

Suwarin Pattamavorakun [*a] and Suwat Pattamavorakun[b]

[a]Science Faculty, [b]Business Administration Faculty
Rajamangala U niversity of Technology, Thanyaburi
Klong Six, Pathumthani, 12110 Thailand

ABSTRACT

In recent years, artificial neural networks have been applied in many fields. Reportedly, the structure of networks seriously affects the performance of the network model. A scheme for river flow forecasting with a recurrent neural network has been proposed in this paper which consists of the phase that determines of input patterns. Autocorrelation analysis is used to identify the number of input nodes of time series for training. The calculated number of neurons for an input layer is then used to construct the fully recurrent neural network forecaster. Computer simulations are presented to show the effectiveness of the scheme. The results obtained show that autocorrelation and cross correlation analysis can be used to determine the number of input nodes. In terms of the performance statistics, an online training algorithm for fully recurrent neural networks has high forecasting accuracy.

KEYWORDS: Autocorrelation analysis, crosscorrelation analysis, network architecture, fully recurrent neural networks, number of input nodes, efficiency index, computational time.

1. INTRODUCTION

River flow forecasting is one of the earliest forecasting problems that have attracted the interest of scientists. The importance of estimating river flows to the livelihoods of the inhabitants around rivers made them study and record their lives since the earliest history. Forcasting the behavior of complex systems has been a broad application domain for neural networks. In particular, applications such as forecasting natural and physical phenomena [1] have been widely studied. Forecasting is a highly "noisy" application, because we are typically dealing with systems that receive thousands of inputs, which interact in a complex nonlinear fashion. Usually with a very small subset of inputs or measurements available from the system, the system behavior is to be estimated and extrapolated into the future. What gives some hope to solve some of the difficult forecasting problems is that typically successive events or inputs that affect the time series are serially correlated, and that cause a time series pattern that can give some hint of the future.

We present here another neural-network forecasting application, namely river flow forecasting. Forecasting river flows is an important application, that attracted the interest of scientists for more than 50 yrs. It can help in predicting agricultural water supply, predicting potential flood damage, estimating loads on bridges, etc. Application of neural networks in time series forecasting [2] is based on the ability of neural networks to approximate nonlinear functions. The most popular treatment of input data is feeding the neural networks with either the data at each observation or the data from several successive observations. This treatment considers the time series as a nonlinear time series and tends to generate a nonlinear "auto-regression" model to fit the series. So far there have been a few papers describing how to choose inputs for the neural network forecaster (especially for recurrent neural network) in order to achieve better forecasting performance (especially for river flow forecasting). It is our belief that the performance of a recurrent neural networks forecaster is much affected by input data patterns. In this study the first goal is to apply fully recurrent neural network (FRNN) to the problem of predicting the flow of two stations, one is in Thailand and the second is in

* Corresponding author. Tel:02-549-4194 Fax:02-549-4195, Tel:02-549-4828 Fax:02-549-3242
E-mail: suwarin@rmut.ac.th, suwat@rmut.ac.th

Vietnam. In addition we exploit the statistics method, autocorrelation and cross correlation analysis to determine the input patterns.

2. THE NETWORKS ARCHITECTURE

An artificial neural network (ANN) is typically composed of layers of nodes. In this study, all the input nodes are in one input layer, all the output nodes are in one output layer and the hidden nodes are distributed into one or more hidden layers in between. In designing an RNN one must determine the following variables:

- the number of input nodes.
- the number of hidden layers and hidden nodes.
- the number of output nodes.

The selection of these parameters is basically problem-dependent. To date, there is no simple clear-cut method for determination of these parameters. Guidelines are either heuristic or based on simulations derived from limited experiments. Hence the design of an ANN is more of an art than a science. The hidden layer and nodes play important roles for many successful applications of neural networks. Influenced by theoretical works which show that a single hidden layer is sufficient for ANNs to approximate any complex nonlinear function with any desired accuracy [2], most authors use only one hidden layer for forecasting purposes.

The number of input nodes corresponds to the number of variables in the input vector used to forecast future values. For causal forecasting, the number of inputs is usually transparent and relatively easy to choose. In a time series forecasting problem, the number of input nodes corresponds to the number of lagged observations used to discover the underlying pattern in a time series and to make forecasts for future values. However, currently there is no suggested systematic way to determine this number. The selection of this parameter should be included in the model construction process. Ideally, we desire a small number of essential nodes which can unveil the unique features embedded in the data. Too few or too many input nodes can affect either the learning or prediction capability of the network. In our opinion, the number of input nodes is probably the most critical decision variable for a time series forecasting problem since it contains the important information about the complex (linear and /or nonlinear) autocorrelation structure in the data. We believe that this parameter can be determined by theoretical research in nonlinear time series analysis and hence improve the neural network model building process. Over the past decade, a number of statistical tests for nonlinear dependencies of time series such as Lagrange multiplier tests [3, 4], likelihood ratio-based test [5], bispectrum tests [6], have been proposed. However, most tests are model dependent and none is superior to others in all situations.

The number of output nodes is relatively easy to specify as it is directly related to the problem under study. For a time series forecasting problem, the number of output nodes often corresponds to the forecasting horizon. There are two types of forecasting: one-step-ahead (which uses one output node) and multi-step forecasts. The first is called the iterative forecasting as used in the Box-Jenkins model in which the forecast values are iteratively used as inputs for the next forecasts. In this case, only one output node is necessary. The second called the direct method is to let the neural network have several output nodes to directly forecast each step into the future.

3. DETERMINATION OF THE NUMBER OF INPUT NODES

Denote the data at instant k as $y(k)$, where y may be a vector, then the above treatment can be described as $y_{k+1} = NN(y_k)$ or $y_{k+1} = NN(y_k, y_{k-1}, \ldots, y_{k-n})$ respectively, where RNN() stands for the neural network forecaster and / is the number of successive observations. This treatment considers the time series as a nonlinear time series and tends to generate a nonlinear "auto-regression" model to fit the series. Autocorrelation analysis has been often used in time series forecasting using statistical approaches such as ARMA models. This analysis is mainly used in determining the autocorrelation between successive observations of time series, and used in the well-known ARIMA models with Box-Jenkins methods that are very efficient in forecasting linear time series [7].

Autocorrelation analysis can be used to determine the correct input patterns for nonlinear time series forecasting with a neural network. This research presents a scheme for time series forecasting with a RNN for detection of input patterns. In the detection phase, autocorrelation analysis is used to

identify input patterns of time series for training. Input pattern detection is done using autocorrelations analysis.

The determination involves two steps:

Step 1 For a given time series, calculate the autocorrelation coefficients. If a trend is detected, then differencing should be used to remove the trend. This step should be repeated until the trend is removed to a reasonable degree. This step is important as autocorrelation coefficients may be used to determine the lags of residuals for short term forecasting.

Step 2 Calculate the partial autocorrelation coefficients. The information will tell us how y_t is auto-correlated to y_{t-k}. Choose those partial autocorrelation coefficients that are significantly different from the rest of the coefficients. The largest lag between any two of the coefficients is the number of inputs needed for the neural network forecaster.

4. EMPIRICAL STUDY

4.1 Network Used

- Fully Recurrent Neural Networks (FRNNs)

FRNNs are complex, dynamical systems, and exhibit all of the power and instability associated with limit cycles and chaotic behavior of such systems. FRNNs can take an in-determinate amount of time. Shown in Figure. 1 is a typical FRNN where all the outputs of the existing neurons are used for feedback and its neurons are fully connected. The simplest memory element is the unit time delay, which has the transfer function, $H(Z) = Z^{-1}$ [8].

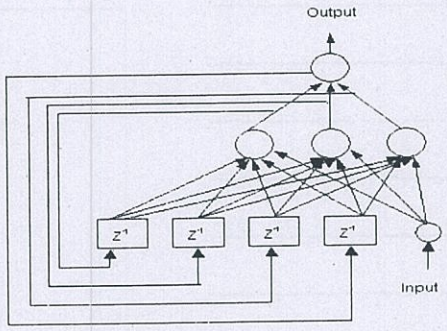

Figure 1. Fully recurrent neural network

4.2 Training Algorithms

- Notation

In an FRNN, m(n)-p-r topology, the total inputs pass directly with their connected weights and then obtain the outputs of both hidden and output nodes simultaneously. Let $y(t)$ or y_t denote the n-tuple of outputs of the units in the network at time t, and let $u(t)$ or u_t denote the m-tuple of external input signals to the network at time t. In a fully-connected RNN, each neuron in both the hidden and output layers have a feedback loop from the neuron itself and the recurrent links from other neurons. To distinguish the components of y representing unit output from those representing external input values, let **O** denote the set of indices i such that y_i is the output of a unit in the network and **I** denote the set of indices i for which y_i is an external input.

$$y_i(t) = \begin{cases} u_i(t), & i \in \mathbf{I} = 1, 2, \ldots, m \\ y_i(t), & i \in \mathbf{O} = 1, 2, \ldots, n \end{cases}$$

Let w_{ij} denote the weight on the connection to the i^{th} unit from either the j^{th} unit, if y either $j \in O$ or if $j \in I$.

Let n = p + r and wij be the weight of link from node j to node i, where,

$i = 1, \ldots, r, r+1, \ldots n = r+p$; and

$j = 1, \ldots, r, r+1, \ldots n = r+p, n+1, \ldots, n+m$.

- Y-N and Atiya-Parlos Algorithm

The important algorithm was derived by Yamamoto and Nikiforuk [9]. The algorithm is derived using an algebraic method instead of a gradient approach. This algorithm, denoted Y-N algorithm, incorporates an Error Back Propagation (EBP) method for obtaining a fictitious target signal of output of the hidden nodes. The weight parameters are obtained by an Exponentially Weighted Least Squares (EWLS) method.

Atiya and Parlos [10] obtained the algorithm by approximating the error gradient. This is an online algorithm which assumes a small change in the network state and finds the direction of weight change by approximation. When minimizing the error, the error of hidden node is equal zero, whereas the error of output node is the difference between desired and model output. The determination of the change in the weights will be according to the change in network state. The basic idea of Atiya- Parlos algorithm is to interchange the roles of the network states $y(t)$ and the weight matrix **W**. The states are considered as the control variables, and the change in weights is determined according to the change in $y(t)$.

Proposed Algorithm

The algorithm was obtained by incorporating the techniques of Y-N algorithm and the error self recurrent (ESR) into the Atiya-Parlos algorithm.

The algorithm finds the direction of weight change by approximation, and estimates fictitious target signals for each hidden node and then uses these signals to adjust output weights, moreover, this algorithm improves the error functions by calculating the errors from output nodes and then feeding back to determine weight updates of output nodes.

The proposed algorithm can be summarized as follows:

1. Assume that the data

   $(y(1), d(1), \ldots, y(T-1), d(T-1))$ are given.

2. Initialize the value of t = 1, and let's denote some mathematical notations in order to find weight changes as follows

$$\gamma(T) = -D^{-1}(T-1)[se(T)]^T + W[se(T-1)]^T \qquad (1)$$

$$B(T) = \sum_{t=1}^{T} \gamma(t) y^T(t-1) \qquad (2)$$

Because of ill-conditioning problems, Eq.(3) must be added with the outer product matrix by a small matrix εI, where ε is a small positive constant.

$$V(T) = \varepsilon I + \sum_{t=0}^{T-1} y(t) y^T(t) \qquad (3)$$

Updated weights of hidden nodes are expressed by

$$\Delta W_H = \eta B_H(T) V_H^{-1}(T) \qquad (4)$$

$$W_H(new) = W_H(old) + \Delta W_H \qquad (5)$$

Then, all new hidden outputs are calculated based on the updated hidden weights.

$$y_H^*(t) = W_H^T(t) y(t-1) \text{ and}$$

$$y^*(t)^T = \left(y_o(t)^T, y_H^*(t)^T, u(t)^T\right) \qquad (6)$$

Finally, updated weights of output nodes are expressed by

$$\Delta W_O = \eta B_O(T) V_O^{-1}(T) \qquad (7)$$

$$W_O(new) = W_O(old) + \Delta W_O \tag{8}$$

3. Calculate a vector of error self-recurrent networks to be used as the error signals for output and hidden nodes

$$\mathbf{se}(t) = \mathbf{e}(t) + \mu_o + \mathbf{e}(t\text{-}1) \tag{9}$$

where $\mathbf{se}(t) = [\mathbf{se}_o(t), \mathbf{se}_h(t)\ldots\ldots, \mathbf{se}_n(t)]$, n-tuples are a number of output units, and μ_o is a constant of the time delayed errors called one-step previous error.

$$e(t) = \begin{bmatrix} x_o(t) - s_o(t) \\ x_h(t) - s_h(t) \end{bmatrix} \tag{10}$$

4. Time step at t = t+1, calculate the model output.

$$y(t) = f(W_y(t\text{-}1)) \tag{11}$$

Then, compute the error signal using $\mathbf{se}(t)$ of the ESR method then using these signal in order to generate $\gamma(t)$ as in Eq.(1).

$$e_i(t) = \begin{cases} y_i(t) - d_i(t), & i \in O \\ 0, & otherwise \end{cases} \tag{12}$$

5. Find weighted changes of hidden nodes. $\Delta W_H(t)$

$$\Delta W_H(t) = \Delta W_H(t-1) + \eta \frac{\gamma_H(t) y^T(t-1) V_H^{-1}(t-1) - B_H(t-1) V_H^{-1}(t-1) y(t-1) \left[V_H^{-1}(t-1) y(t-1) \right]^T}{1 + y^T(t-1) V_H^{-1}(t-1) y(t-1)} \tag{13}$$

and

$$V_H^{-1}(t) = V_H^{-1}(t-1) - \frac{\left[V_H^{-1}(t-1) y(t-1) \right] \left[V_H^{-1}(t-1) y(t-1) \right]^T}{1 + y^T(t-1) V_H^{-1}(t-1) y(t-1)} \tag{14}$$

$$B_H(t) = B_H(t-1) + \gamma_H(t) y^T(t-1) \tag{15}$$

6. All new hidden output signals are calculated based on the updated hidden weights.

$$y_H^*(t) = W_H^T(t) y(t-1) \text{ and } y^*(t)^T = \left(y_o(t)^T, y_H^*(t)^T, u(t)^T \right) \tag{16}$$

7. Find weighted changes of output nodes. $\Delta W_O(t)$

$$\Delta W_o(t) = \Delta W_o(t-1) + \eta \frac{\gamma_o(t) y^*(t-1)^T V_o^{-1}(t-1) - B_o(t-1) V_o^{-1}(t-1) y^*(t-1) \left[V_o^{-1}(t-1) y^*(t-1) \right]^T}{1 + y^*(t-1)^T V_o^{-1}(t-1) y^*(t-1)} \tag{17}$$

and

$$V_O^{-1}(t) = V_O^{-1}(t-1) - \frac{\left[V_O^{-1}(t-1)y^*(t-1)\right]\left[V_O^{-1}(t-1)y^*(t-1)\right]^T}{1+y^*(t-1)^T V_O^{-1}(t-1)y^*(t-1)} \quad (18)$$

$$B_O(t) = B_O(t-1) + \gamma_O(t)y^*(t-1)^T \quad (19)$$

8. Repeat steps 3 through 7 until the end of given data. The training is terminated when the system error has reached an acceptable criterion. Set ΔW_H and ΔW_O to zero at the beginning of next iteration.

4.3 Determination of the number of input nodes

The data used consist of the daily rainfall and stream flow records at two stations, namely Khao Laem in the Mae Klong River Basin Thailand, and Hoa Binh station in the Red River Basin of Vietnam. The data at Khao Laem station between April 1982 and March 1985 are used in the calibration(training) phase, and from April 1985 to March 1986 for validation (testing) stage. For Hoa Binh station, the training data comprise those values during January 1992- December 1994 and the testing data are during January 1995 – December 1995.

The input of the RNN consist of discharge and rainfall, and their values for this forecasting problem are used

$$Q(t+\tau) = g(Q(t), Q(t-1).....Q(t-N_Q), R(t), R(t-1).....R(t-N_R))$$

where g stands for "function of", Q(t) and R(t) denote the discharge and rainfall at time t respectively, N_Q and N_R are the lag times in the discharge and rainfall data respectively, and τ is the forecasting lead time.

The SPSS soft ware (standard version 11.0 for Windows 2000) was used to calculate the autocorrelation function (ACF), partial autocorrelation function (PACF) and cross correlation function (CCF) between successive observation of discharges and rainfalls. The autocorrelation analysis is used to determine the number of input node [11] for the same output variable. Similarly, the cross-correlation analysis can also be used to determine the number of input nodes for other variables. However, in practice, these are used as the upper limit for the number of nodes representing the different input variables.

- **Khao Laem data**

The ACF and PACF of the discharge with lags from 1 to 365 are shown in Figure 2 and 3 respectively.

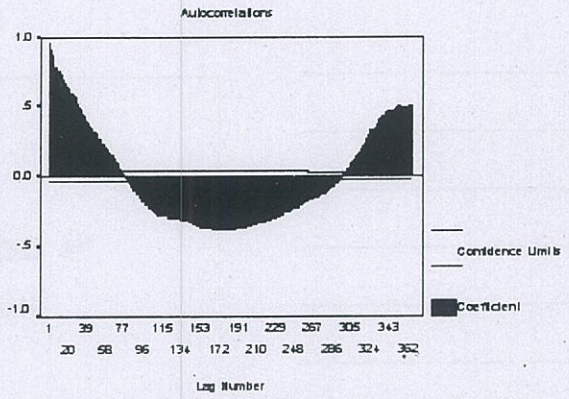

Figure 2. ACF of the discharge data from 1 to 365

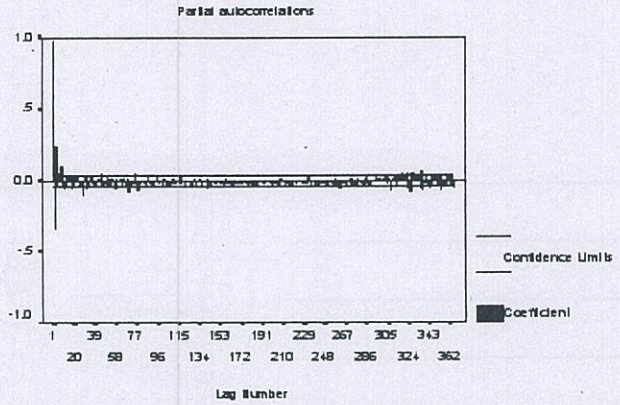

Figure 3. The partial autocorrelations of the discharge from 1 to 365

The discharge data series was non-stationary in the mean (the mean of these series changes over time), therefore the data was applied in the first differencing : $y'_t = y_t - y_{t-1}$. Figures 4 and 5 clearly show that the non-stationary has been effectively removed.

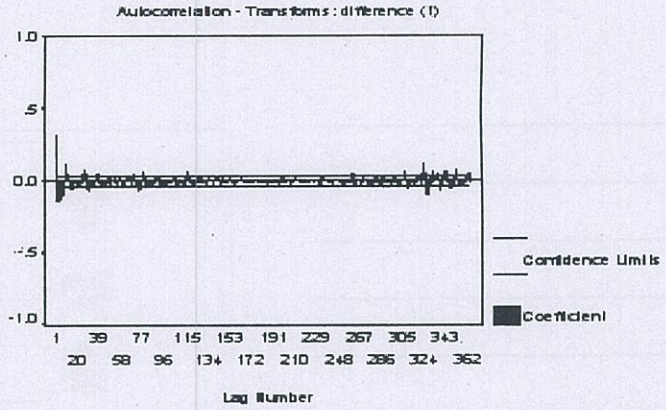

Figure 4. ACF of the discharge data from 1 to 365 with the first difference

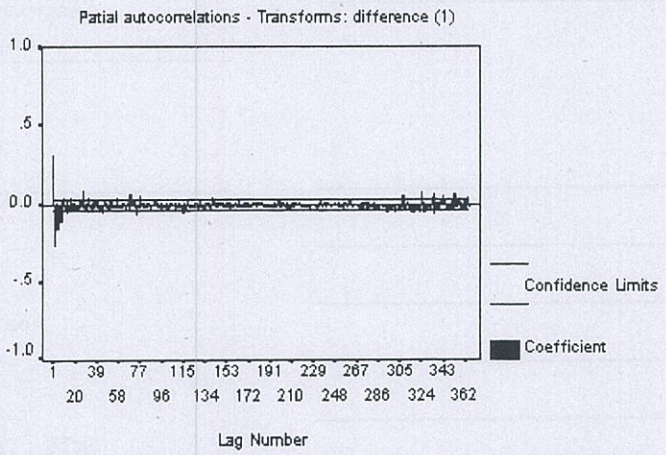

Figure 5. PACF of the discharge data from 1 to 365 with the first difference

Similarly, the cross correlation coefficient function (CCF) of discharge and rainfall data with lags from 1 to 20 was computed. The CCF is non-stationary in the mean, therefore, the first differencing was taken and the result is in Figure 7.

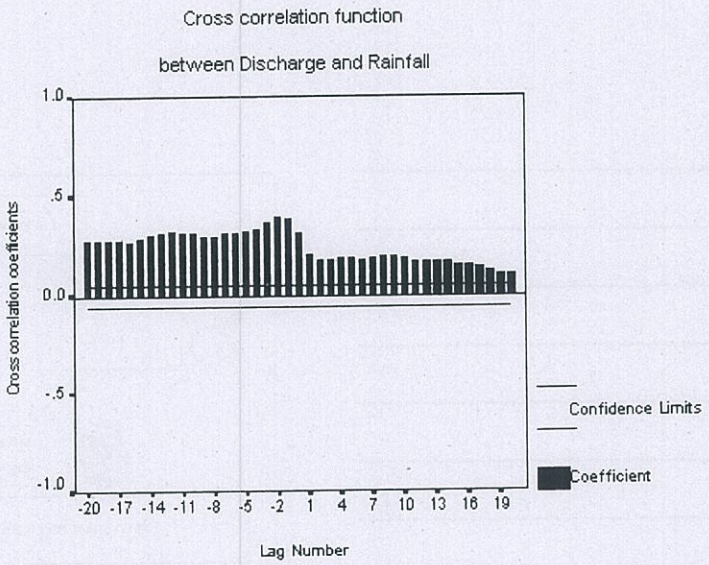

Figure 6. Cross correlations between discharge and rainfall

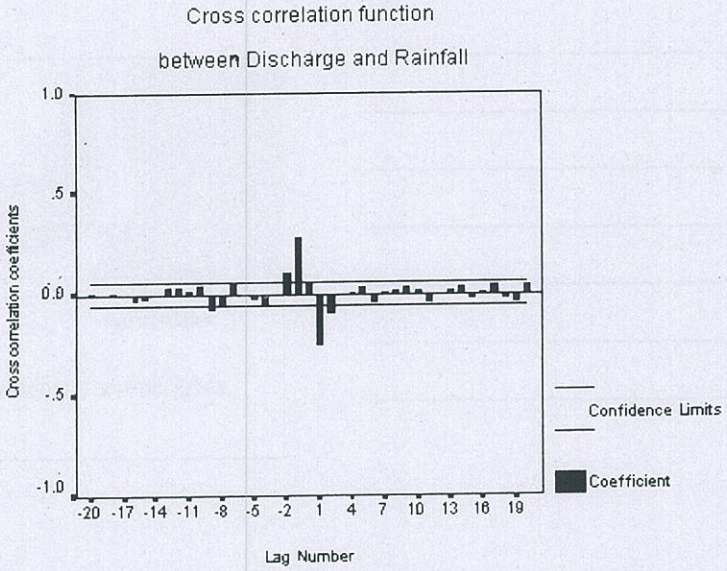

Figure 7. Cross correlations between discharge and rainfall with the first difference

The partial autocorrelation of discharge data (y_k). Figure 8 depict that PACF between y_k and y_{k+i}, i = 1, 2....., 16. Considering the sampling distribution of autocorrelation coefficients, r_k: for k = 1.....16 of the time series. So the PACF between y_k and y_{k+1}, y_k and y_{k+2} and y_k and y_{k+3} are significantly different from others.

In addition, the cross correlation values in Figure 9 depict that the CCF between discharge (y_k) and rainfall (R_k) data, y_k and R_k, y_{k-1} and R_{k-1} and y_{k-2} and R_{k-2} are significantly different from others. Thus, the RNN forecaster without differencing has six inputs y_k, y_{k-1}, y_{k-2}, R_k, R_{k-1}, R_{k-2} where y_k is the discharge and R_k is the rainfall at time k. In other words, the discharge at time t+1 is a function of discharge values at time t, t-1 and t-2 and of rainfall values at times t, t-1 and t-2:

$$Q(t+1) = g(Q_t, Q_{t-1}, Q_{t-2}, R_t, R_{t-1}, R_{t-2})$$

Partial Autocorrelation: KLM
Transformation: natural log, difference (1)

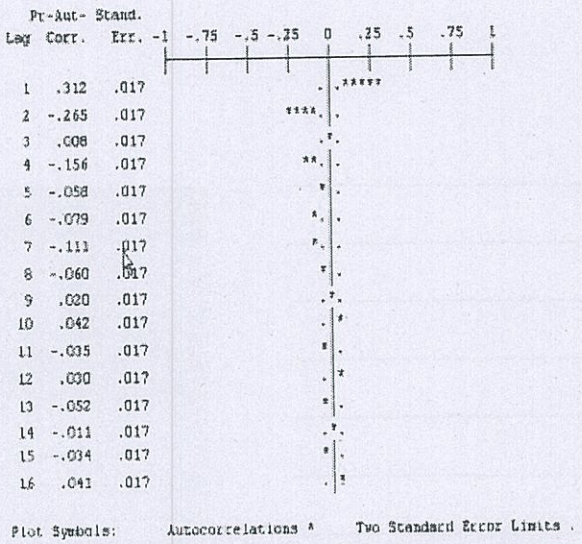

Figure 8. Computable first lags after differencing: 3283

Cross Correlations: KLM discharge Rainfall
Transformations: difference (1)

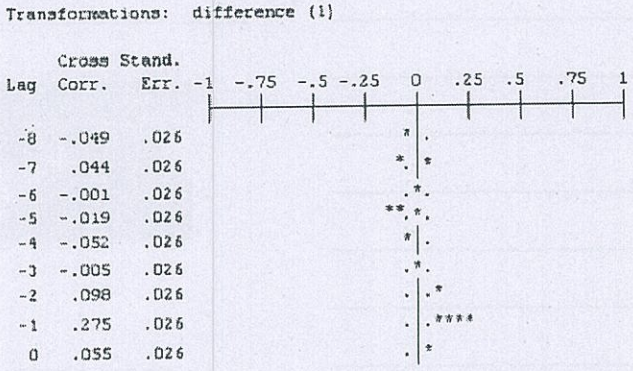

Figure 9. Computable 0-order correlations after differencing : 3284

- **Hoa Binh data**

Similarly, with the method for determining the number of hidden node of discharge and rainfall data points of the Khao Laem data was also applied for the Hoa Binh data.

In this study, the hydrological variables of Hoa Binh data are also the discharge and rainfall and due to quite the same size (considered rather big sizes) of these two drainage area of the streamflow station used, so we obtained the same number of input nodes of Hoa Binh data as Khao Laem data.

4.4 Data Used

Following the work of Danh et al. [12], the forecasting equation with lead time of one day can be expressed as:

$$Q(t+1) = g(Q(t), Q(t-1), Q(t-2), R(t), R(t-1), R(t-2)) + h(Q_{H1}(k-1), \ldots Q_{Hp}(k-1), Q_o(k-1))$$

t = daily data time (Sundays to Fridays)
k = time step of the network
Q = discharge (m^3/s) at the forecast station
R = average rainfall (mm/d) at the station used to forecast
H_i = hidden node i (i = 1, 2,….p).
o = output node

Before being presented to the RNN, the data are transformed by an affine transformation to the range [0.05, 0.95] using the equations:

$$y'_t = [0.9(y_t - y_{min})/(y_{max} - y_{min})] + 0.05 \quad (20)$$

$$y_t = [(y_{max} - y_{min})(y'_t - 0.05)/0.9] + y_{min} \quad (21)$$

where y'_t and y_t denote the transformed and original data, and y_{max} and y_{min} are the maximum and minimum of the original data in the calibration phase, respectively.

A stopping rule is introduced to indicate the convergence of the algorithm. It is based on the relationship between the summation errors of two consecutive iterations. Generally, the stopping rule is as follows:

$$\left| \frac{SE(t+1) - SE(t)}{SE(t)} \right| \leq \varepsilon_l$$

where SE(t+1) is the Sum of Squared Errors at iteration (t+1), and ε_l is a small number. The value of ε_l is chosen to be 0.001.

- Networks Used

For both data sets, the 6(3)-2-1 topology is used for fully recurrent network (FRNN). This topology has 6 external inputs (3 inputs of discharge and 3 inputs of rainfall data), 3 feedback lines from hidden nodes and 1 output node. To obtain the performance of the training algorithms, the weights in the range of [0, 1] were used. We note that for recurrent networks, it is always better to start with small weights, because if we have long sequences for large initial weights the states tend to wander off into the saturation region.

All feedback connection weights equal 1.0. The sigmoid slope (ρ) of sigmoid function was set to 1. The training parameters, for the proposed algorithm was set the bias of one-step previous error (μ_0) to be 0.3, η is equal to 0.008 and ε is equal to 6.0.

A prediction accuracy measure is often defined in terms of the forecasting error which is the difference between the actual (desired) and the predicted values. Nash and Sutcliffee [13] proposed the **efficiency index (EI)** as follows:

$$EI = SR/ST$$

where SR = ST-SE

$$ST = \sum_{i=1}^{M} (y_i - \bar{y})^2$$

$$SE = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2$$

$$\bar{y} = \frac{1}{M} \sum_{i=1}^{M} y_i$$

In the above equations,

SR = Variation explained by the model.

ST = Total variation.

SE = Sum of squared errors.

$\hat{y}_i$ = Actual output, i.e. observed value at time

$\bar{y}$ = Mean value of the actual output.

y_i = Model output, i.e. forecast value at time

M = Number of data points

From the definition equation, EI is clearly related to SE: the minimum value of SE corresponds to the maximum value of EI. Moreover, a value of EI close to 1.0 indicates an excellent

performance in terms of the prediction accuracy. As such, EI can indicate the performance in terms of prediction accuracy of the resulting (network) model.

5. SIMULATION RESULT AND DISCUSSION

(a) Forecasting Accuracy and Computational Time

The calculated values of the efficiency index are collected in Tables 1.

- It should be noted that the computed values of the efficiency index are provided for both stages, training and testing. Generally speaking, the network being considered can be said to provide a good forecasting model if the values of the efficiency index in both stages are high, but more emphasis should be placed upon the training phase. Since the two catchment areas concerned are quite big , the accuracy obtained expressed by the efficiency index (EI) should not be less than 0.85 [12].

Table 1. Computed values of efficiency index and computation time (second)

Station	Training		Testing
	EI	Time	EI
Khao Laem	0.901	8.642	0.905
Hoa Binh	0.914	20.151	0.936

From Table 1. It is clear that the training time is quite small, the highest computation time required 21 seconds for three years of daily data, about 1095 daily values (for Khao Laem and Hoa Binh data) on a Pentium III Processor. For FRNN, the proposed algorithm provides accurate forecast (EI >0.85).

6. CONCLUSIONS

The experiment study was mainly for obtaining the performance of the proposed algorithm on FRNN using the efficiency index (for forecasting model accuracy) and the computation time. For a given problem, the network builder needs to experiment to find the appropriate one for the problem at hand. For this river flow forecasting using discharge and rainfall data in the Khao Laem station, Thailand and Hoa Binh station, Vietnam, the number of input nodes is well defined by the autocorrelation and cross correlation analysis and it is the number of independent variables associated with the problem. By means of the experiments using hydrological data, that such network structures provide the resulting model which have the good performance in terms of efficiency index and low computation time.

7. ACKNOWLEDGEMENTS

The authors would like to acknowledge Assistant Dr. Jurairat Duangdeun of Faculty of Science, Rajamangala University of Technology, Thanyaburi (RMUT) for a kind encouragement throughout the study, and for providing the opportunity to submit this paper to KSAS.

REFERENCES

[1] Weigend, A. and Gerschenfield, N. **1994** Eds. *Time Serie, Prediction : Forecasting the Future and Understanding the Past*. Reading, MA, Addison-Wesley.

[2] Cybenko, G. **1989** Approximation by Superpositions of Sigmoidal Function, *Mathe-matical Control System*. 2, 303-314.

[3] Luukkonen, R., Saikkonen, P. and Terasirta, T. **1988** Testing Linearity in Univariate Time Series Models, *Scandinavian Journal of Statistics* 15, 161-175.

[4] Saikkonen, P. and Luukonen, R. **1988** Lagrange Multiplier Tests for Testing Non-Linearities in Time Series Models, *Scandinavian Journal of Statistics* 15, 55-68.

[5] Chan, W.S. and Tong, H. **1986** On Tests for Non-Linearity in Time Series Analysis, *Journal of Forecasting* 5, 217-228.

[6] Hinich, M.J. **1982** Testing for Gaussianity and Linearity of a Stationary Time Series, *Journal of Time Series Analysis*. 3, 169-176.

[7] Jarrett, J. **1991** *Business Forecasting Methods*. Basil Black Well.

[8] William, R and Zipser, D. **1989** A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*. 1, 270.

[9] Yamamoto, Y. and Nikiforuk, P.N. **1999** A Learning Algorithm for Recurrent Neural Networks and Its Application to Nonlinear Identification. *Proceeding of the 1999 IEEE International Symposium on Computer Aided Control System Design*. Hawaii. pp. 551-556.

[10] Atiya, A.F. El-Shoura, S.M. and Shaheen, S.I. **1999** Comparison Between Neural Network Forecasting Techniques-Case Study : River Flow Forecasting. *IEEE Transaction on Neural Networks*. 10(2). 402-409.

[11] Lin, F. Tu, X.H. Gregos, S. and Irons, R. **1995** Time Series Forecasting with Neural Networks. *Complexity International*. 2, 1-12.

[12] Danh, N.T. Phien, H.N. and Gupta, A.D. **1999** Neural Network Models for River Flow Forecasting. *Water SA*. 25(1). 33-39.

[13] Nash, J.E. and Sutchffee, J.V. **1970** River Flow Forecasting Through Conceptual Models. *Journal of Hydrology*. 10, 282-290.