8. ADAPTIVE NEURAL NETWORKS IN REGULATION OF RIVER FLOWS

J. MOHAN REDDY
Department of Civil Engineering
Colorado State University
Fort Collins, Colorado 80523

BOGDAN M. WILAMOWSKI
Department of Electrical Engineering
University of Wyoming
Laramie, WY 82071

8.1 Problem Statement

The demand for water is growing as a result of population growth, competition from agricultural and industrial sectors, global warming, and pollution of water resources. Judicious utilization and conservation of the available water resources is of paramount importance in order to meet the growing demand for water. One of the ways to conserve water is to estimate the water demand accurately, and provide just the right quantity of water to the users, i.e. match supply with demand as closely as possible.

Generally, water is released from a reservoir in response to an anticipated and/or known demand in the command area of a water resources project. When the distances between the source and the delivery points along the river reach are very long, the demand must be known several hours to several days in advance so that water can be released in time for the users to take full advantage of it. Furthermore, the amount of water released from a reservoir is not the amount that reaches the downstream delivery points some time later. Many things can happen between the upstream and downstream points of a river reach and affect the flow rate. Everything involved in the hydrologic cycle and the hydro-geologic properties of the land adjacent to the stream can play a major role in the actual loss or gain of water along a river reach. Some of the factors that influence the loss/gain of a river reach are: the length of the reach, natural flow of the river, size of increase in flow, precipitation, elevation and slope of water table, evaporation, evapotranspiration, stream channel characteristics, silt layer characteristics, hydraulic characteristics of the aquifer, irrigation return flows, diversions, and valley cross-sections.

For accurate computation of releases from a reservoir or flow rate at an upstream point along a river reach, in order to meet downstream water demands, several different types of river modeling algorithms can be used. The Saint-Venant equations of open-channel flow or its variants can be conveniently used to model flow through river reaches, with excellent agreement between the numerical integration of the complete hydrodynamic equations and field measurements. Recently, the USGS (United States Geological Survey) has developed a computer program for numerical modeling of floods through a river network. However, this approach has some practical limitations, particularly when applied to long river reaches, because obtaining accurate data on river cross-sections, roughness values, seepage losses, and bed slope at several points along a river reach is a tedious task. Therefore, simplified methods that are reasonably accurate but do not need extensive information on reach characteristics have been developed. One of the most widely used methods of river flow modeling is called storage routing. This process involves computing the change in storage, *ΔS*, in the river reach for a given time increment. The term 'routing' generally refers to the accounting of water as it travels through a channel. Storage routing methods use the continuity equation in its integral form that is given as follows:

$$I\Delta t - \Delta S = O\Delta t \tag{8.1}$$

where *I* and *O* = average rates of inflow and outflow, respectively, for the time interval *Δt;* and *ΔS* = change in volume (storage) of water in the channel reach between the inflow point and the outflow point during the time interval *Δt* (Franzini et. al 1992). Given the inflow as a function of time, there are two unknowns in Eq. 8.1. Therefore, one more equation, to replace the momentum equation of open-channel flow, is needed to solve for the outflow from the reach. McCarthy (1938) suggested an approach in which the storage S is assumed to be represented by a linear relation of inflow and outflow. In mathematical terms, the storage and the continuity equations are given as follows:

$$S_t = K[\alpha I_t + (1-\alpha)O_t] \tag{8.2}$$

$$dS_t/dt = I_t - O_t \tag{8.3}$$

in which S_t, I_t and O_t represent the simultaneous storage, inflow, and outflow, respectively, at time t; K = storage-time constant for the river reach, which has a value reasonably close to the flow travel time of the river reach; and α is a weighting factor. McCarthy (1938) used the above method for flood control studies in the Muskingum River basin in Ohio. Equation 8.2 forms the basis for the Muskingum routing method, which is an example of the simplest and most frequently used form of the routing models (Kraijenhoff and Moll, 1986).

To solve for the outflow from a river reach when the inflow is known, Eqs. 8.2 and 8.3 are combined. By writing the combined equation for time t and t+1, and solving for outflow at time t+1 results in the following:

$$O_{t+1} = C_0 I_{t+1} + C_1 I_t + C_2 O_t \tag{8.4}$$

in which the coefficients are given as

$$C_0 = \frac{0.5\Delta t - K\alpha}{K(1-\alpha) + 0.5\Delta t} \tag{8.5}$$

$$C_1 = \frac{0.5\Delta t + K\alpha}{K(1-\alpha) + 0.5\Delta t} \tag{8.6}$$

$$C_2 = \frac{K(1-\alpha) - 0.5\Delta t}{K(1-\alpha) + 0.5\Delta t} \tag{8.7}$$

where K and α are estimated using past records of inflow and outflow, and are assumed to be the characteristic values for the reach. These parameters are estimated using a trial-and-error procedure which is time consuming and prone to subjective interpretation. Since natural streams are continually changing and the phenomenon is nonlinear, the accuracy of the above method is not very good. Over the years, several approximate methods have been applied to the above equation. Cunge (1969) proposed a modified version of the Muskingum model, called the Muskingum-Cunge model, which is of the following form:

$$Q_{i+1}(t+1) = C_1 Q_i(t) + C_2 Q_i(t+1) + C_3 Q_{i+1}(t) + C_4 \qquad (8.8)$$

in which Q denotes the discharge, subscript i denotes the upstream end of the routing reach, subscript i+1 denotes the downstream end of the routing reach, and time instants t and t+1 are Δt time units apart. The coefficients C_1, C_2, and C_3 are calculated using the observed inflow and outflow data, and other kinematic characteristics of the river reach, and C_4 is related to lateral inflow/outflow from the river reach. Using a finite-difference approximation to the spatio-temporal variation of flow rate, Cunge (1969) derived expressions for the above coefficients that are very similar to the coefficients presented in Eqs. 8.5 to 8.7. In fact, under some assumptions, the numerical values of the coefficients in Eqs. 8.4 and 8.8 would be identical (Montes, 1998), except for the coefficient C_4.

Recently, Papageorgious and Messmer (1989) proposed an optimal feedback control algorithm for regulating flow in river reaches. Though the method is straight forward, since the coefficients of the control algorithm are not adaptive to the changing conditions in the river reach, the performance of the control algorithm was not satisfactory as the fluctuations in flow through the river reach became large.

8.2 ANN Approach

8.2.1 DRAWBACKS OF EARLIER METHODS

Ideally, the flow rate released from reservoirs, computed using the assumed travel time, lateral inflows and outflows, and conveyance losses in the reach, must result in a discharge rate that is equal to the required discharge rates at all the delivery points along the river reach. However, because of the inaccuracies in the travel time of the reach, the estimated conveyance losses, and unexpected flows from rain or snowmelt, etc., the delivered discharge would normally differ from the required discharge rates or demand. Water users are put in difficult situations when the actual discharges are less than the requested amount, and, conversely, the delivered water may not be beneficially used when the amount delivered is more than the amount requested. In order to improve users' confidence in the water delivery system and to encourage efficient use (water conservation) of water delivered to the users, there should be a "close match" between the amount of water requested and the amount of water delivered. Unfortunately, this "close match" is difficult to achieve with the techniques mentioned above because some of the parameters used in the movement of water from reservoirs to the point of use are not accurate and are variable during any given period (Georgakakos, et. al 1990). Hence, improved methodologies based on real-time (on-line) acquisition of stream flow data are needed (Reddy, 1997). Georgakakos et. al (1990) have proposed a state-space version of the Muskingum-Cunge routing model for real-time flood forecasting. The form of the equations is as follows:

$$\begin{aligned} Q(t+1) &= AQ(t) + BU(t) + Cq(t) + w(t) \\ z(t) &= H^T Q(t) + v(t) \end{aligned} \tag{8.9}$$

in which A,B, and C = system constants based upon the geometry and the reference flow condition; z = measured outflow rate; q = inflow/outflow rate from the reach; w = random input noise acting on the system; and v = random measurement noise. Kalman filtering technique (Brown and Hwang, 1997) along with real-time measurements was used for flood forecasting purposes. Though this technique can account for some degree of uncertainty in the estimation of the values for the system parameters, it will not be able to adapt the system parameters to the changing conditions in the river reaches. Therefore, an adaptive system identification scheme is needed for developing an appropriate river flow regulation algorithm.

8.2.2 POTENTIAL OF ANNs

A neural network is a massively parallel distributed processor that is designed to model, in a primitive fashion, the functioning of a brain; hence the name Artificial Neural Network (ANN). Neural networks have a natural propensity for storing experiential knowledge and making it available for use (Haykin, 1999; Anderson, 1995). ANNs can be placed into one of three classes based on their feedback link connection structure: recurrent structure (global feedback connections), local recurrent structure (local feedback connections, e.g., cellular neural networks), and non-recurrent (no feedback connections). A special type of non-recurrent ANN is the feedforward neural network, which is used in this paper.

With the advent of fast data communication technology, remote data acquisition and control is being used widely in several industries, including water resources systems. Advanced mathematical techniques, combined with real-time data acquisition systems, provide numerous possibilities for controlling complex, distributed, and uncertain dynamic systems, particularly in situations where a good model of the system is not available, and an approximate model of the system must be identified using on-line data. Artificial neural networks can provide the user with a tool to accomplish this and, though not a new concept, they have become increasingly popular due to the speed of modern computers as they require many calculations during the training process. Narendra and Parthasarathy (1990), Kuschewski et. al (1993), Wilamowski (1996), and Levin and Narendra (1996) used artificial neural networks for simultaneous identification and control of dynamic systems. The application of artificial neural networks to problems in water resources systems is gaining momentum because of its success in dealing with complex problems.

ANN consists of layers of neurons with synaptic (weighted) links connecting the outputs of neurons in one layer to the inputs of neurons in the next layer. Neural networks basically consist of several neurons which are the basic information processing units. The three basic elements of a neuron are explained below.

Synapses or connecting links: These are characterized by their strength (or weight) of its own, and are used to weigh the input received from a sensor in producing the specified output from the system. The weight is positive if the associated input is excitary; it is negative if the input is inhibitory. A signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. The first subscript refers to the neuron in question, and the second subscript refers to the input end of the synapse to which the weight refers.

Adder: The net effect of all the inputs on a given neuron (output) is obtained by summing the products of the synaptic weights and the associated input strengths. This is basically a linear operation, and is given by the following expression:

$$u_j = \sum_{i=1}^{p} w_{ji} x_i \tag{8.10}$$

in which u_j = net effect of all the inputs on neuron k; w_{ji} = synaptic weight connecting input i with neuron j; x_i = input variable i; and p = number of inputs in the problem.

Activation function: This function is used to limit the amplitude of the output of a neuron. Multi-layer neural networks usually use continuous activation functions, either unipolar

$$y = \varphi(u) = \frac{1}{1 + \exp(-\lambda u)} \tag{8.11}$$

or bipolar

$$y = \varphi(u) = \tanh(0.5\lambda u) = \frac{2}{1 + \exp(-\lambda u)} - 1 \tag{8.12}$$

where λ= shape factor. These continuous activation functions allow for the gradient-based training of multi-layer networks. Typical activation functions are shown in Figure 8.1.

The simplest and most commonly used neural networks use only one directional signal flow. Furthermore, most of the feedforward neural networks are organized in layers. An example of a three-layered feedforward neural network is shown in Figure 8.2. This network consists of input nodes, two *hidden layers*, and an output layer.

The feedforward neural networks are used for nonlinear transformation (mapping) of a multidimensional input variable into another multidimensional output variable. In theory, any input-output mapping should be possible if the neural network has enough neurons in the hidden layers (size of output layer is set by the number of outputs required).

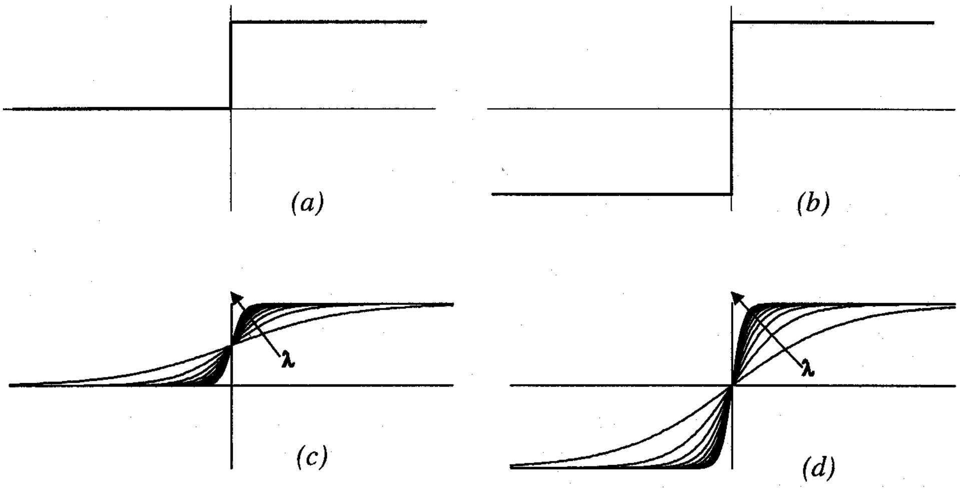


Figure 8.1. Typical activation functions: *(a)* hard threshold unipolar, *(b)* hard threshold bipolar, *(c)* continuous unipolar, *(d)* continuous bipolar.

Practically, it is not an easy task and, presently, there is no satisfactory method to define how many neurons should be used in hidden layers. Usually this is found by trial and error. In general, it is known that if more neurons are used, more complicated shapes can be mapped. Conversely, networks with large number of neurons lose their ability for generalization, and it is more likely that such network will also try to map noise supplied to the input (Haykin 1994).

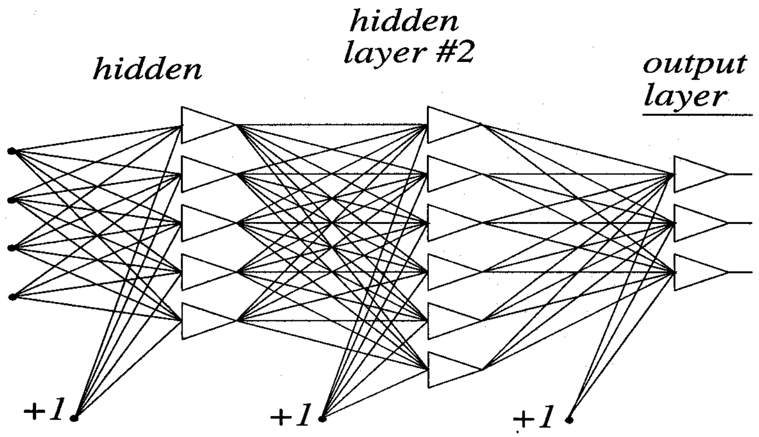


Figure 8.2. An example of the three-layered feedforward neural network, which is also known as the back-propagation network.

Weights in artificial neurons are adjusted during a training procedure. Various learning algorithms were developed but only a few are suitable for multi-layer neuron networks. Some use only local information about signals in the neurons whereas others require information from outputs. Supervised algorithms require a supervisor (known outputs) who always knows what the outputs should be whereas unsupervised algorithms need no such information. Though there are several well know learning rules for training neural networks, the most commonly used learning method for multi-layered feedforward networks is the Error Back Propagation (EBP) method or a higher-order variant of this method such as the Levenberg–Marquardt algorithm. The Error Back Propagation algorithm is described first.

Standard Error Back Propagation Algorithm

In a multi-layered feedforward algorithm, the error from output neuron j at time instant n is given by

$$e_j(n) = \hat{y}_j - y_j \tag{8.13}$$

in which e_j = error at output neuron j; $\hat{y}_j$ = predicted output at neuron j; and y_j =actual output at neuron j. The instantaneous sum of squared errors of the network is given as:

$$E(n) = 1/2 \sum_{j \varepsilon C} e_j^2(n) \tag{8.14}$$

where the set C includes all the neurons in the output layer of the network. If N is the number of examples in the training set, then the average squared error is obtained by summing the error over the entire training set and normalizing the error with respect to the set size N. In training the neural network, the error is utilized to compute the incremental changes in the synaptic weights of the neural network. In the standard back-propagation algorithm, the weight increments are computed using the following expression:

$$\Delta w_{ji} = \eta \delta_j(n) x_i(n) \tag{8.15}$$

in which η = learning rate constant; and $\delta_j(n)$ = local error-gradient. For the output neuron j, it is given as follows:

$$\delta_j(n) = e_j(n) \varphi_j'(v_j(n)) \tag{8.16}$$

$$\delta_j(n) = -\frac{\partial E(n) \partial e_j(n) \partial \hat{y}_j(n)}{\partial e_j(n) \partial \hat{y}_j(n) \partial v_j(n)} \tag{8.17}$$

For a hidden neuron j, the error gradient is given as follows:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{8.18}$$

Once an activation function is selected, the above procedure is straightforward. However, the main issue is with the selection of a value for the learning rate constant, η. A large value for the learning rate results in faster convergence but, in some cases, it might result in unstable system behavior. Conversely, a smaller rate of learning constant results in stable but very slow learning. The selection of an appropriate value for the learning rate constant, therefore, is a trial and error procedure. Also, the original back-propagation algorithm was found to have a tendency for oscillation. Recently, there have been some modifications to the above method (Rumelhart et. al 1986) which are expected to result in faster convergence. In order to smooth the process, the increment of weights Δw_{ij} can be modified as follows:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + (1-\alpha)\eta \delta_j(n) x_i(n) \tag{8.19}$$

in which α = momentum coefficient.

The back-propagation algorithm consists of two distinct computational passes. In the forward pass, given the set of inputs at time n, the network computes the outputs from all the neurons in the hidden layer(s) in sequence. Then, using the output from the last hidden layer, the network computes the output from all the neurons in the output layer. In the computation of the output from a neuron, it uses the equation of the selected activation function for the given neuron. Usually, the same activation function is used for all the neurons in a given layer. By comparing the computed output with the target output from each of the neurons in the output layer, the sum of squared errors from the network is computed. This ends the forward pass of the algorithm.

During the backward pass, the error gradients at each neuron of each layer in the network are computed, which in turn are used to compute the incremental changes in the synaptic weights at time n. This concludes the backward pass in which the output error is propagated backwards and used to modify the synaptic weights. New values for the synaptic weights are computed, and the procedure is repeated with the next set of input-output data.

The back-propagation algorithm can be significantly speeded up, when after finding the components of the gradient, the weights are modified along the gradient direction until a minimum is reached. This process can be carried on without the necessity of computationally intensive gradient calculation at each step. The new gradient components are calculated once a minimum on the direction of the previous gradient is obtained. This process is only possible for cumulative weight adjustment. One method to find a minimum along the gradient direction is the three-step process of finding error for three points along the gradient direction and then, using a parabolic approximation, jump directly to the minimum. The fast learning algorithm using the above approach was proposed by Fahlman (1989) and is known as the *quickprop*.

The back-propagation algorithm has many disadvantages, which lead to very slow convergence. One of the most debilitating is the fact that, in the back-propagation algorithm, the learning process almost perishes for neurons responding with the maximally wrong answer. For example if the value on the neuron output is close to $+1$ and desired output should be close to -1, then the neuron gain $\varphi(u)$ and the error signal cannot back propagate, so the learning procedure is not effective. To overcome this difficulty, a modified method for derivative calculation was introduced by Wilamowski and Torvik (1993). The derivative is calculated as the slope of a line connecting the point of the output value with the point of the desired value as shown in Figure 8.3.

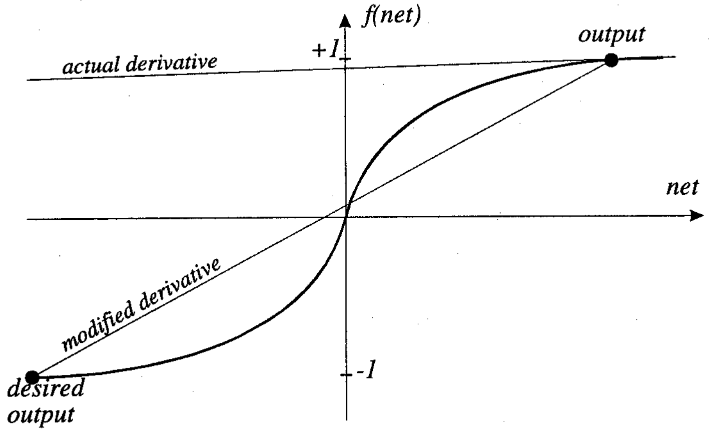


Figure 8.3. Illustration of the modified derivative calculation for faster convergence of the error back-propagation algorithm.

$$\varphi_{modif} = \frac{y_{desired} - y_{actual}}{u_{desired} - u_{actual}} \tag{8.20}$$

Note that for small errors, Eq. 20 converges to the derivative of activation function at the point of the output value. With an increase in the system dimensionality, a chance for local minima decreases. It is believed that the phenomenon described above, rather than a trapping in local minima, is responsible for convergence problems in the error back-propagation algorithm.

Levenberg-Marquardt (LM) method

The Levenberg-Marquardt learning algorithm (Hagan and Menhaj, 1994) is a second-order search method of a minimum. At each iteration step, the error surface is represented by parabolic approximation and the minimum of the paraboloid is the solution for the step.

Simple approaches require function approximation by the first term of the Taylor series

$$F(\mathbf{w}_{k+1}) = F(\mathbf{w}_k + \Delta\mathbf{w}) + \mathbf{g}_k^T \Delta\mathbf{w}_k + \frac{1}{2}\Delta\mathbf{w}_k^T \mathbf{A}_k \Delta\mathbf{w}_k \tag{8.21}$$

where $\mathbf{g} = \nabla E$ is error gradient; and $\mathbf{A} = \nabla^2 E$ is the Hessian of the global error E. The gradient and the Hessian are computed as follows:

$$Gradient \;\Rightarrow\; \frac{\partial E}{\partial w_1},\; \frac{\partial E}{\partial w_2},\; \frac{\partial E}{\partial w_3},\; \cdots$$

$$Hessian \Rightarrow \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \frac{\partial^2 E}{\partial w_1 \partial w_3} & \cdots \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \frac{\partial^2 E}{\partial w_2 \partial w_3} & \cdots \\ \frac{\partial^2 E}{\partial w_3 \partial w_1} & \frac{\partial^2 E}{\partial w_3 \partial w_2} & \frac{\partial^2 E}{\partial w_3^2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The steepest decent (error back-propagation) method calculates weights using:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha\, \mathbf{g} \tag{8.22}$$

while the Newton method uses:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mathbf{A}_k^{-1}\mathbf{g} \tag{8.23}$$

The Newton method is practical only for small networks where Hessian $\mathbf{A}_k$ can be calculated and inverted. In the Levenberg-Marquardt method the Hessian $\mathbf{A}_k$ is approximated by product of Jacobians

$$\mathbf{A} \approx 2\mathbf{J}^T\mathbf{J} \tag{8.24}$$

and gradient as

$$\mathbf{g} \approx 2\mathbf{J}^T\mathbf{e} \tag{8.25}$$

where $\mathbf{e}$ = vector of output errors; and the Jacobian $\mathbf{J}$ is computed using

$$\mathbf{J} = \begin{bmatrix} \frac{\partial e_{11}}{\partial w_1} & \frac{\partial e_{11}}{\partial w_2} & \cdots & \frac{\partial e_{11}}{\partial w_N} \\ \frac{\partial e_{21}}{\partial w_1} & \frac{\partial e_{21}}{\partial w_2} & \cdots & \frac{\partial e_{21}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{K1}}{\partial w_1} & \frac{\partial e_{K1}}{\partial w_2} & \cdots & \frac{\partial e_{K1}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{1P}}{\partial w_1} & \frac{\partial e_{1P}}{\partial w_2} & \cdots & \frac{\partial e_{1P}}{\partial w_N} \\ \frac{\partial e_{2P}}{\partial w_1} & \frac{\partial e_{2P}}{\partial w_2} & \cdots & \frac{\partial e_{2P}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{KP}}{\partial w_1} & \frac{\partial e_{KP}}{\partial w_2} & \cdots & \frac{\partial e_{KP}}{\partial w_N} \end{bmatrix} \tag{8.26}$$

It is much easier to calculate the Jacobian than the Hessian and also usually the Jacobian is much smaller so less memory is required. Therefore weights can be calculated as

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(2\mathbf{J}_k^T\mathbf{J}_k\right)^{-1} 2\mathbf{J}_k^T\mathbf{e} \tag{8.27}$$

or

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}_k^T \mathbf{J}_k\right)^{-1} \mathbf{J}_k^T \mathbf{e} \tag{8.28}$$

To ensure convergence, the Levenberg-Marquardt algorithm introduces the μ parameter

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}_k^T \mathbf{J}_k + \mu \mathbf{I}\right)^{-1} \mathbf{J}_k^T \mathbf{e} \tag{8.29}$$

where **I** is identity unit matrix, μ is a learning parameter and **J** is Jacobian of m output errors with respect to n weights of the neural network. For $\mu = 0$ it becomes the Gauss-Newton method. For very large μ the LM algorithm becomes the steepest decent or the EBP algorithm. The μ parameter is automatically adjusted during computation process so that good convergence is secured. The LM algorithm requires computation of the Jacobian **J** matrix at each iteration step and the inversion of $\mathbf{J}^T\mathbf{J}$ square matrix. Note that in the LM algorithm an N by N matrix must be inverted in each iteration. The Levenberg-Marquardt recently became very popular because it usually converges in 5 to 10 iterations. The main drawback of this method is that it requires the storage of some matrices which can be quite large for certain problems.

Cascade-Correlation architecture

The cascade correlation architecture was proposed by Fahlman and Lebiere (1990). The process of network building starts with a one layer neural network, and hidden neurons are added as needed. The network architecture is shown in Figure 4.8.4.

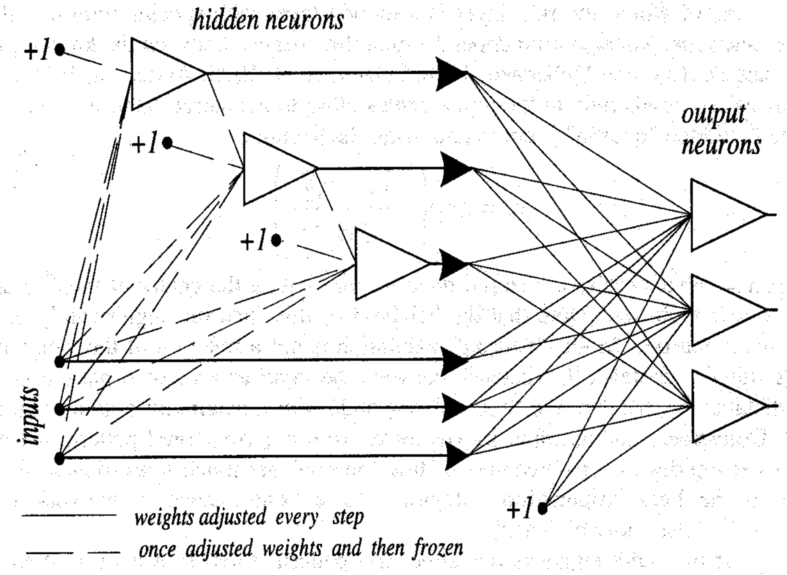

hidden neurons
+1
+1
+1
output neurons
outputs
inputs
+1
weights adjusted every step
once adjusted weights and then frozen

Figure 8.4. The cascade-correlation architecture.

In each training step, a new hidden neuron is added and its weights are adjusted to maximize the magnitude of the correlation between the new hidden neuron output and the residual error signal on the network output. In the process, the correlation parameter, *S*, defined below must be maximized

$$S = \sum_{o=1}^{O} \left| \sum_{p=1}^{P} \left(V_p - \overline{V}\right)\left(E_{po} - \overline{E_o}\right) \right| \tag{8.30}$$

where O = number of network outputs; P = number of training patterns; V_p = output on the new hidden neuron; and E_{po} = error on the network output. By finding the gradient, $\Delta S/\Delta w_i$, the weight adjustment for the new neuron is found as

$$\Delta w_i = \sum_{o=1}^{O} \sum_{p=1}^{P} \sigma_o \left(E_{po} - \overline{E_o}\right) f_{p'}\ x_{ip} \tag{8.31}$$

where σ_o = sign of the correlation between the new neuron output value and network output; $f_p{}'$ = derivative of the activation function for pattern p; and x_{ip} = input signal. The output neurons are trained using the delta (back-propagation) algorithm. Each hidden neuron is trained just once and then its weights are frozen. The network learning is deemed complete and the architecture finalized when satisfactory agreement is obtained between network predictions and target outputs.

Radial basis function networks

The structure of the radial basis network is shown in Figure 8.5. This network consists of two layers, of which the first layer is a hidden layer with special neurons called the radial basis functions, whose output depends upon the distance between the known pattern and the new pattern (Luo and Unbehaen, 1998; Hagen, et. al, 1995; Anderson, 1995). Each of these "neurons" responds only to the input signals close to the stored pattern. The output signal y_i of the i^{th} hidden "neuron" is computed using the formula

$$y_i = \exp\left(- \frac{\left\| x - s_i \right\|^2}{2\sigma_i^{\ 2}} \right) \tag{8.32}$$

where x = input vector; s_i = stored pattern representing the center of the i^{th} cluster; and σ_i = radius of this cluster. Note that the behavior of this "neuron" significantly differs form the biological neuron. In this "neuron", excitation is not a function of the weighted sum of the input signals. Instead, the distance between the input and a stored pattern is computed. If this distance is zero then the "neuron" responds with a maximum output magnitude equal to one. Conversely, as the input moves away from a given stored pattern, the neuron output drops off rapidly to zero. Features of this "neuron" are much more powerful than a neuron used in the back-propagation networks. As a consequence, a network made of such "neurons" is also more powerful.

If the input signal is the same as a pattern stored in a neuron, then this "neuron" responds with *1* and remaining "neurons" have *0* on the output, as illustrated in Figure 8.5. Thus, output signals are exactly equal to the weights coming out from the active "neuron". This way, if the number of "neurons" in the hidden layer is large, then any input output mapping can be obtained. Unfortunately, it may also happen that for some patterns several "neurons" in the first layer will respond with a non-zero signal. For a proper approximation the sum of all signals from hidden layer should be equal to one. In order to meet this requirement, the output signals are often normalized as shown in Figure 8.5.

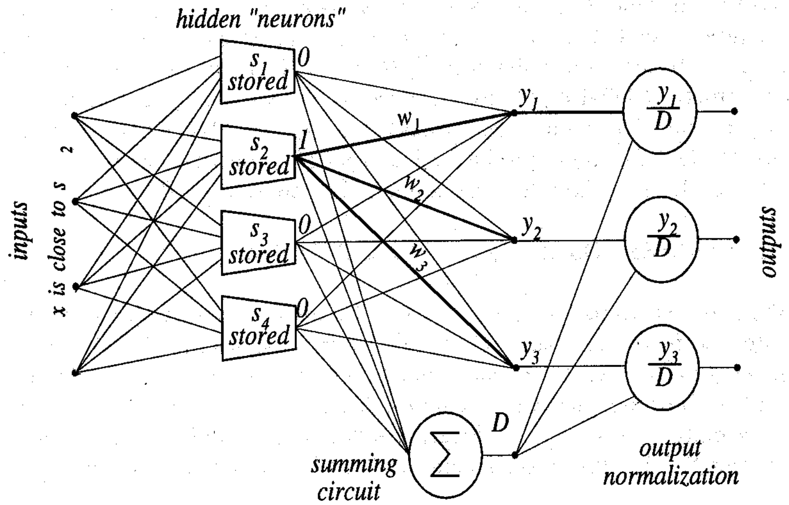


Figure 8.5. Typical structure of a radial basis function network.

The radial based networks could be designed or trained. Training is usually carried out in two steps. In the first step the hidden layer is usually trained in the unsupervised mode for choosing best patterns for cluster representation. Also in this step, radii σ_i are found using the following relationship to account for proper overlapping of clusters:

$$\sigma_i = \frac{1}{p}\sum_{j=1}^{p}\left\| s_j - s_i \right\| \tag{8.33}$$

where p = number of nearest vectors to s_i; and s_j = one of the p nearest neighbors of the center vector s_i. The second step of training is the error back-propagation algorithm carried out only for the second (or output) layer. The second layer is linear and produces a weighted sum of the outputs of the first (hidden) layer. Since this is a supervised algorithm for one layer only, the training is very rapid, 100 or 1000 times faster than the error back-propagation based multi-layer networks. This makes the radial basis function network very attractive. A more detailed analysis of radial basis functions can be found in Chapter 5.

8.2.3 REAL-TIME DATA ACQUISITION SYSTEMS

The main objective of an adaptive control systems is to adjust the coefficients of the control equation as the system behavior changes (evolves) over-time. In order to adjust the control equation coefficients, real-time information on the system behavior is needed.

This information must be available to the neural network model on a continuous basis. Either a dedicated remote data acquisition system or an Internet-based data acquisition system can be used to acquire the data on a continuous basis. The acquired data can then be imported into the desired computational software used for system identification and control. In order to demonstrate the technique, the existing USGS stream flow database was accessed from their Web site. A special software (C language based), that builds upon the Windows Sockets program, was developed to access the required stream flow data from the web site. Since the Neural Network Toolbox of The Mathworks, Inc. (Beale and Demuth, 1992-1997), was proposed to be used for system identification and control, the special C program was compiled using the MATLAB's External (mex) applications interface facility to produce a *.dll file. This file was then used to acquire the data into MATLAB using the following command at the MATLAB prompt:

```
>> data=readusbr('ARKCOSAL','Q','98APR25,98APR26')
```

in which the first term in the brackets indicates the station identification code, the second term indicates the flow rates data, and the third term indicates the date(s) of interest. The USGS has designed the real-time data acquisition system with a sampling interval of 15-minutes, and a reporting interval of 4-hours. This program worked very well for our research. Currently, commercial software packages are available for importing data directly into MATLAB, either from a dedicated or a web based data acquisition system.

8.2.4 ON-LINE SYSTEM IDENTIFICATION AND ADAPTIVE REGULATION OF RIVER FLOWS

Several hardware/software solutions are currently available to interface real-time data with mathematical models. Here, the existing USGS/USBR stream flow database was accessed using a custom designed software. The data were imported directly into the neural network software. In order to save storage space, stream flow data that were more than one year old were stored in the form of average flow rate per day rather than instantaneous readings at every 15 minutes. Therefore, for testing the methodology, the daily average flow rate data were used for system identification purposes. The available data were always split into two sets, and one set was used to train the neural network whereas the other set was used to test the performance of the trained network model. This analysis was done in three river basins - the Green River basin, the North Platte River basin, and the Arkansas River basin. Here, the analyses done on the North Platte river and the Arkansas river are discussed. If historical data do not exist, or if the interest is to perform on-line identification, a neural network can be trained on a continuous basis using the time-series data of inflow into and outflow from the given reach. Therefore, on-line system identification involves training of a neural network using the most current data.

For the purpose of system identification, the input data consisted of the inflow into the river reach at the upstream point, and all the other known lateral inflows/outflows between the two end-points of the reach. Sometimes, these inflows may not be known accurately. Under these conditions, at least an approximate estimation of the inflow, that is equal to the statistical average, may be included for training the network. Conversely, if the magnitudes of the inflows/outflows are not known but the magnitude remains more or less constant, then the inflow/outflow value can be completely ignored in training the network, because the effect of the constant inflow/outflow would be absorbed by the weights of the neural network.

In the case of adaptive regulation of river flows, the objective is to estimate the desired inflow into the reach, given the downstream demands and the inflow/outflow rates along the river reach. Therefore, the outflow rate at the downstream-end of the river reach along with the inflow and outflow rates from the reach are considered inputs to the neural network, whereas the inflow time sequence into the river reach becomes the neural network output. Of course, the information on approximate travel-time along the river reach must be available. Once the neural network is trained to predict the required inflow rate into the river reach in order to meet the downstream demand, then for any other given downstream flow rate demand the required upstream flow rate can be computed using the neural network model of the system. When the computed inflow rate is input to the neural network, the actual outflow from the reach must be very close to the target (downstream demand) flow rate. The above approach assumes that the system parameters do not change significantly over the time period of interest. In reality, the river reach dynamics might change quite rapidly depending upon rainfall in the basin or withdrawal by water users. A large difference between the target demand and the actual outflow rate indicates that the system parameters have changed. Then the neural network must be trained using the most current data set to capture the latest system dynamics, i.e. adapt the system model coefficients, in this case the synaptic weights and biases of the neural network. Since we did not have the authorization to change the inflow rate into the river reaches, i.e. the releases from the reservoirs, we did not use the last approach to do an adaptive regulation of river flows.

8.3 Application of ANN to River Flow Regulation

8.3.1 APPLICATION OF ANN TO WATER RESOURCES PROBLEMS

There has been a rapid growth in the application of ANN to identification and control of systems during the last two decades. Recently, this technique has found several applications in the field of water resources (French et. al, 1992; Mason, et. al, 1996; Mohan, 1995; Raman and Chandramouli, 1996; Reddy, 1995,1997; Karunanithi, et. al, 1994; Dandy et. al, 1996; DeSilets, et. al, 1992; Kao, 1996; Ranjithan, et. al, 1993).

The prerequisite to the application of artificial neural networks technique is the availability of sufficient amount of data for the system under consideration. Once the sets of input and output data have been identified and acquired, several algorithms such as the Widrow-Hoff algorithm (Etxebarria, 1994), the radial basis function networks (Mason et. al 1996), and the multi-layered feedforward networks (Karunanithi et. al 1994; French et. al 1996) are available to train neural network models. The Widrow-Hoff algorithm basically consists of a single-layer network with a linear activation function. There is no significant advantage to using single layered neural networks because their structure is similar to a multiple, linear regression algorithm (Warwick, 1995). Here, the multi-layered, feedforward neural network and the radial basis function network were selected for application.

8.3.2 APPLICATION TO NORTH PLATTE RIVER IN WYOMING

The North Platte River, near the Wyoming-Nebraska border, was used as the first test case. Figure 8.6 is a map of the area. The upstream station is just below the Whalen Diversion Dam. The Whalen Diversion Dam is where two major irrigation canals begin. The downstream station is at the Wyoming-Nebraska border. The length of the reach is about 45 miles with several incoming tributaries along the reach. During peak runoff, the Laramie River contributes significantly to the North Platte and since there is a stream flow gauge located near the confluence, this data was included in the model as additional information.

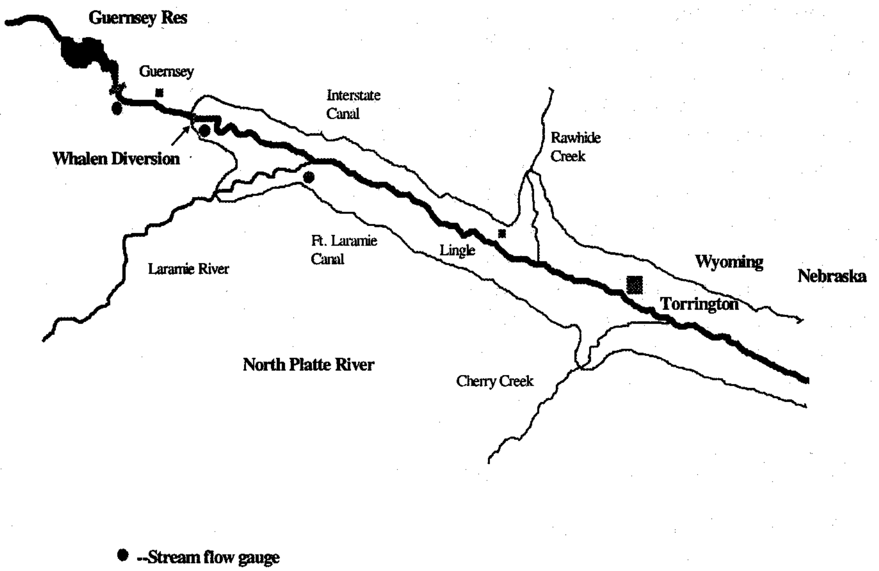


Figure 8.6. Schematic of the North Platte River System

Data for several days from the Wyoming-Nebraska border station, again encompassing the lag-time of the reach, and from the Laramie River station were used as inputs to the network. A two-layered, feedforward network with error back-propagation was used. In order to speed up the computation time, the Levenberg-Marquardt algorithm was invoked. The neural network was trained or calibrated using data of one season (May-September, 1994) and then tested on the data of the next season (May-September, 1995). Figure 8.7 is a plot of the network output shown in dotted line. The solid line is the target data, which was the actual data at the Whalen Diversion station.

This type of scenario is a very real problem. There is a desired (pre-specified) demand at the Wyoming-Nebraska border and, by law, Nebraska has a certain right to a percentage of the North Platte water. If the Whalen Diversion Dam was a full scale reservoir instead of simply a diversion dam, this model could either provide information on how much water to release from the reservoir to adequately meet the downstream demand at the border, or the model could be easily changed to give the required release at the Guernsey Reservoir if stream flow information on the two canals that the Whalen Diversion Dam diverts to were known and implemented into the model. In the latter case, inputs would include demand at the border and the demand for each canal. The output would be the release at the Guernsey Reservoir.

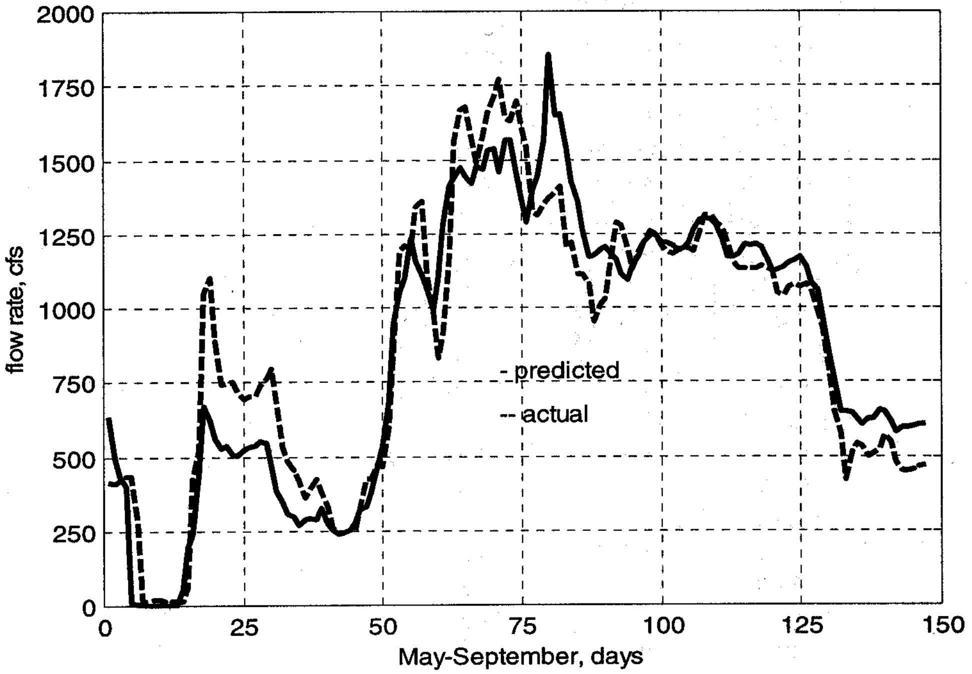
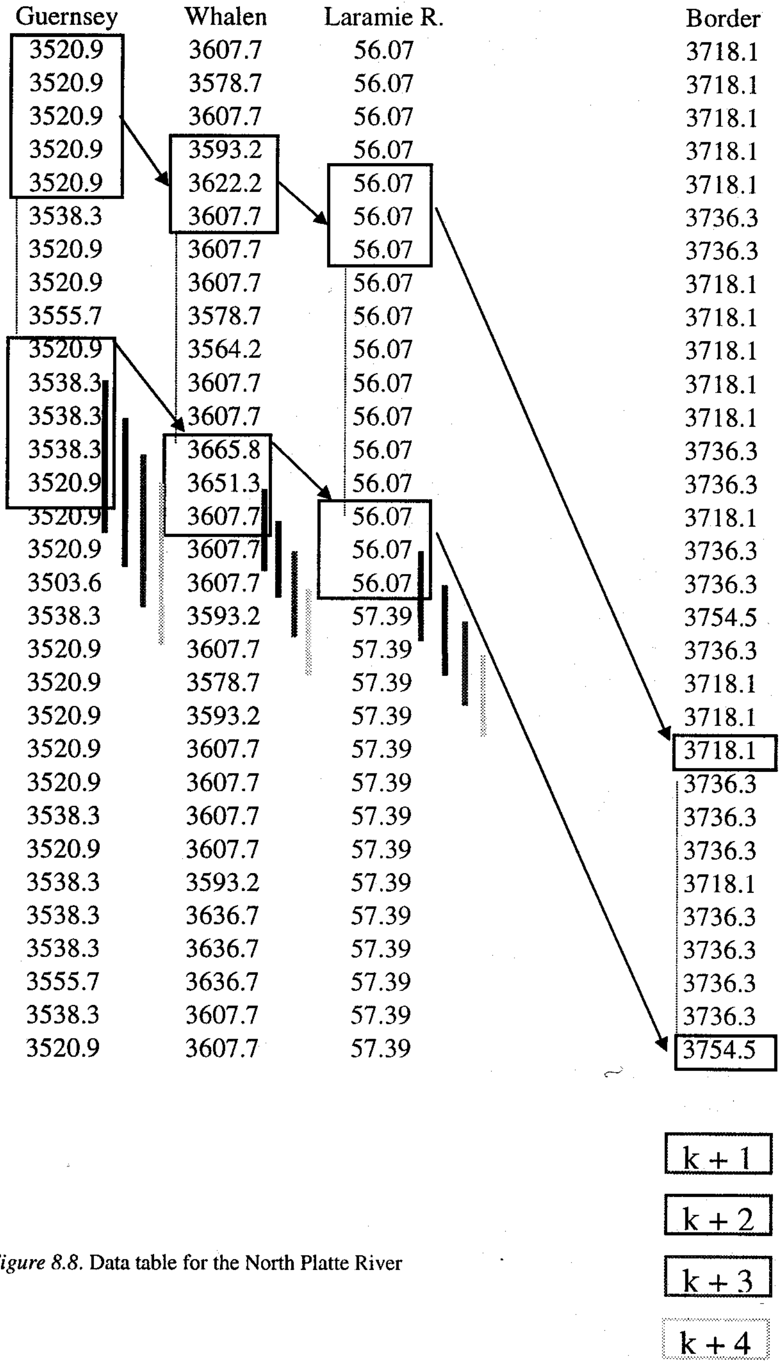


Figure 8.7. Results from MATLAB program for the North Platte River

A more conclusive test was performed using the North Platte River between Guernsey Dam and the Wyoming- Nebraska state line. Figure 8.8 shows the data similar to the way the m-file 'data' would retrieve it. A total of eleven inputs were used to create the neural network model of the North Platte for this particular reach - five values from the station immediately below the Guernsey dam, three values from the Whalen Diversion station, and three values from the incoming Laramie River. Additional inputs were used for this test because there were two significant irrigation canals that started at the Whalen Diversion dam and information here would help in making the downstream predictions. Also, the Laramie River could contribute significantly at times to downstream water quantities and information there could improve downstream predictions during those times. Notice in Figure 8.8 that the input blocks (boxed) were taken at different times relative to the target outflow values (also boxed).



Guernsey	Whalen	Laramie R.	Border
3520.9	3607.7	56.07	3718.1
3520.9	3578.7	56.07	3718.1
3520.9	3607.7	56.07	3718.1
3520.9	3593.2	56.07	3718.1
3520.9	3622.2	56.07	3718.1
3538.3	3607.7	56.07	3736.3
3520.9	3607.7	56.07	3736.3
3520.9	3607.7	56.07	3718.1
3555.7	3578.7	56.07	3718.1
3520.9	3564.2	56.07	3718.1
3538.3	3607.7	56.07	3718.1
3538.3	3607.7	56.07	3718.1
3538.3	3665.8	56.07	3736.3
3520.9	3651.3	56.07	3736.3
3520.9	3607.7	56.07	3718.1
3520.9	3607.7	56.07	3736.3
3503.6	3607.7	56.07	3736.3
3538.3	3593.2	57.39	3754.5
3520.9	3607.7	57.39	3736.3
3520.9	3578.7	57.39	3718.1
3520.9	3593.2	57.39	3718.1
3520.9	3607.7	57.39	3718.1
3520.9	3607.7	57.39	3736.3
3538.3	3607.7	57.39	3736.3
3520.9	3607.7	57.39	3736.3
3538.3	3593.2	57.39	3718.1
3538.3	3636.7	57.39	3736.3
3538.3	3636.7	57.39	3736.3
3555.7	3636.7	57.39	3736.3
3538.3	3607.7	57.39	3736.3
3520.9	3607.7	57.39	3754.5

Figure 8.8. Data table for the North Platte River

on the far right column). The time from the center of each input block to the target outflow value was the lag-time from that station to the stateline station. The lag-time for this particular reach was well over four hours; so, if desired, the program could make predictions for up to twelve hours into the future. However, more accurate results are obtained by using the most recent data. This is done by predicting only four hours into the future, waiting during those four hours to receive new data from the Internet, and then using this most recent data to make the next four hours' predictions (Figure 8.9).

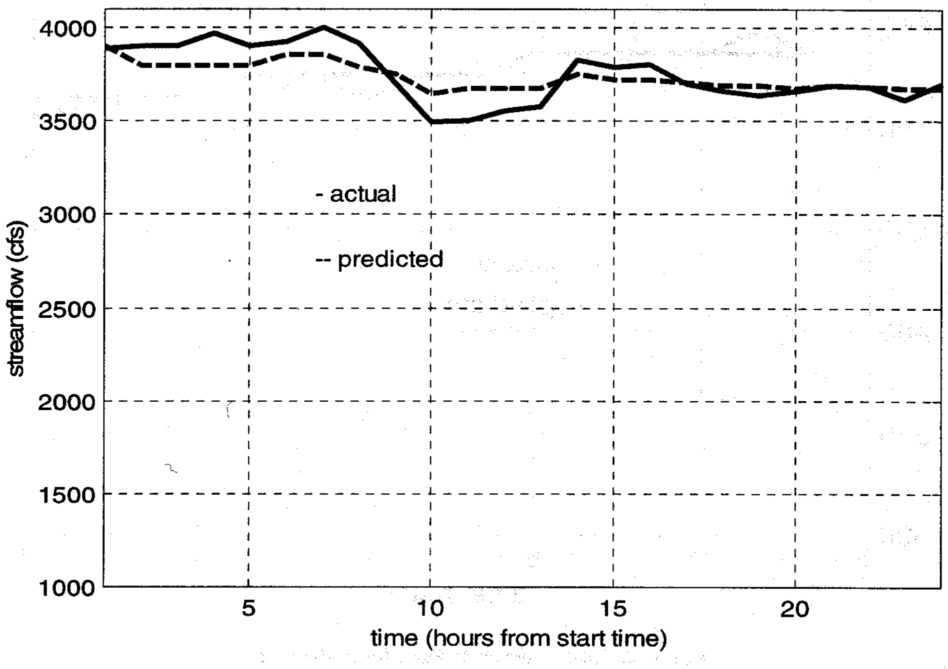


Figure 8.9. Results from North Platte River 25-hour test

The radial basis function network with a gamma type basis function was also applied to the North Platt river data. However, this analysis was done on real-time data that was acquired during the month of April 1998, when the flow in the river reach was low (less than 600 cfs). This neural network performed as well as the one with the back-propagation algorithm (Figure 8.9), and was much faster always (Figure 8.10).

8.3.3 APPLICATION TO ARKANSAS RIVER IN COLORADO

The Arkansas River was chosen for this application because of the warmer climate in Southern Colorado. The time of year for which this first test was performed was February and March. Due to the extreme cold weather in Wyoming, many Wyoming stations were iced up and not in working order until spring. In addition, the stream flows in Wyoming are very low without much change over time, which makes it difficult to tell if the model is really working or not. The reach selected is located between two existing stations that are relatively close together, about 15 miles. One station is near Salida, Colorado, and the other is downstream near Wellsville, Colorado (Figure 8.11).

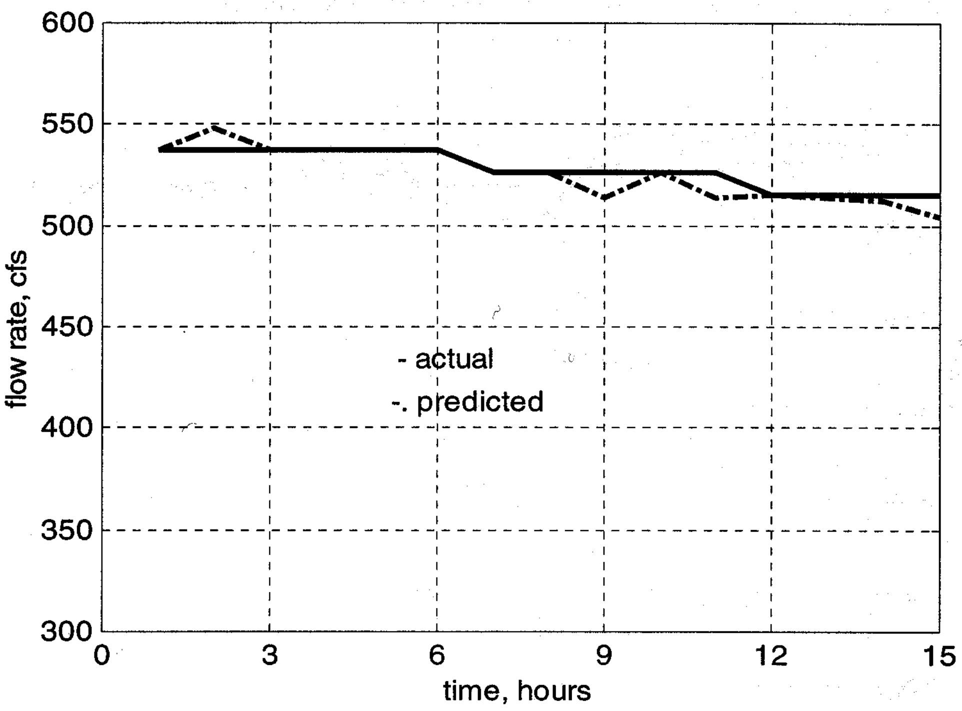


Figure 8.10. Results from North Platte River using RBF

Figure 8.12 displays a sample of data retrieved from the Internet for the two Arkansas River stations. This figure indicates how the data were used in the model. The model was designed to take three values from the upstream station as inputs to a neural network and train the weights and biases so the network output would match the data from the downstream station. Referring to Figure 8.12, the black box around the three values in the left column are the inputs and the network output was trained so the output matches the outlined value in the right column. There were six data sets used for the training of the network. The first set is contained within the black boxes, connected with the arrow and ends, following the dashed line, with the lower data set outlined with the black box. Once the network is trained, three of the most current input data sets remain. Each vertical line represents an input data set. The model takes each data set, applies the trained neural network and produces a predicted output at times k + 1, k + 2 and k + 3.

The bold lines are the inputs for the corresponding output denoted by the same colored box.

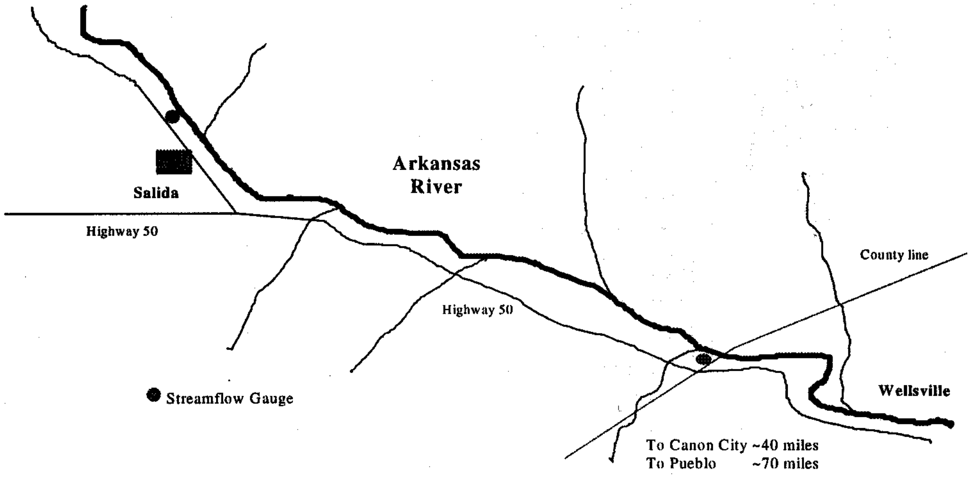


Figure 8.11. Schematic of Arkansas River near Salida, Colorado.

Of the two stations, the upstream data were used as inputs to predict the downstream flow. The main stimulus for this project was the need to more accurately meet downstream demands, meaning the model should make predictions at the release or upstream end of a reach. However, for testing purposes, this model makes downstream predictions given real-time data at the upstream end. The reason for this is that once the prediction is made, the user can wait for the next time increment containing new data and have a target to compare the results against. For example, referring again to Figure 8.12, two columns of hourly data come into the computer every four hours. Three flow rates from the upstream station near Salida at approximately the lag-time of the reach prior to time t were used for inputs to the model. The output was a prediction of the flow rate at the downstream station near Wellsville at time t + 1. After waiting for four hours, since that is how often the USBR updated their web site, the actual flow rate at Wellsville at that time was available for comparison to the model output.

Once again, the custom software developed fetched the data (through the Internet) from the USBR web site and then sorted and organized the data. There are some inconsistencies in the raw data when it comes from the USBR web site. For example, the raw data sometimes repeats a time and stream flow. Also, the raw data provides a stream flow for every fifteen minutes and the m-file data manipulates the raw data and prepares inflow and outflow matrices that contain one stream flow for each hour for the last ten hours from the time the data was retrieved from the Internet.

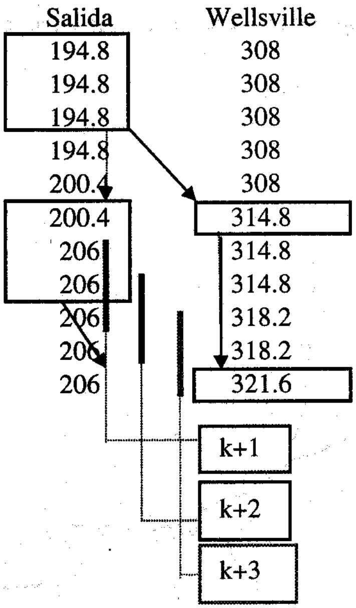


Figure 8.12. Example Stream flows for Arkansas River

Ideally, the program would make predictions for four hours into the future since new data is available only every four hours. But as can be seen in the program, there is only a prediction for the next three hours. The reason for this is that the lag-time of the reach is well under four hours. In order to achieve accurate results from a neural network, the input needs to encompass the lag-time. Referring to Figure 8.12, the time from within the three data values used as inputs to the network to the predicted output value must be close to the lag-time of the reach. On this particular reach, notice that after the last value on the right is used for the training process, there is only enough remaining data values on the left to be used as inputs to the neural network to make three predictions (Figure 8.13). To get good use out of this type of technique on a short reach, new data must be available at time intervals less than the lag-time of the reach. When this model was applied to the North Platte River, this was not a problem since the lag-time of the reach used there was approximately 20 hours.

8.4 Conclusions

The existing methods of river flow regulation do not account for system losses/gains very accurately because of the spatio-temporal variability of the parameters that influence loss/gain from river reaches. To correctly account for the losses, continuous monitoring of the inflow and outflow rate from a selected river reach is required. However, monitoring alone does not help in achieving a close match between the supply and demand at a given point along a river reach. Appropriate regulation schemes that take into account the dynamic travel time and the losses/gains along the selected river reach are desired.

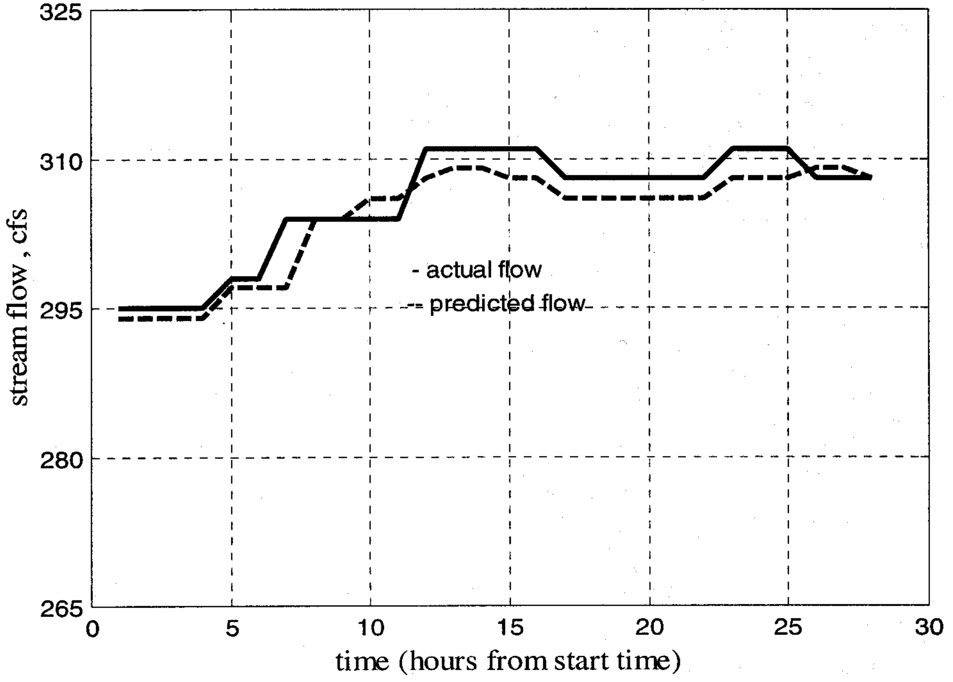


Figure 8.13. Results from Arkansas River test

Artificial neural networks along with real-time data acquisition systems provide a convenient mechanism for on-line system identification and regulation of river flows. Here, two different kinds of neural networks architectures- error back propagation and the radial basis function- were applied to selected reaches of the North Platt river in Wyoming, and the Arkansas river in Colorado. In both the cases, the performance of the neural network models was found to be good in terms of matching the supply with demand.

References

Anderson, J.A. (1995). *An Introduction to Neural Networks*, MIT Press, Cambridge, MA.

Beale, M. and Demuth, H. (1992-1997) *MATLAB Neural Network ToolBox User's Guide*, The Mathworks, Inc., Natick, MA.

Brown, R.G. and Hwang,P.Y.C. (1997) *Introduction to Random Signals and Applied Kalman Filtering*. John Wiley & Sons, New York, NY.

Cunge, J.A. (1969) On the subject of a flood propagation computation method (Muskingum Method). *Journal of Hydraulic Research*, 7(2) : 205-230.

Dandy, G.C., Simpson, A.R. and Murphy, L.J. (1996) An improved genetic algorithm for pipe network optimization. *Water Resources Research*, 32(2), 449-458.

DeSilets, L., Golden, B., Wang, Q. and Kumar, R. (1992). Predicting salinity in Chesapeake Bay using back-propagation, *Computers and Operations Research*, 19(3/4), 277-285.

Etxebarria, V. (1994) Adaptive control of discrete-time systems using neural networks, *IEEE Proceedings on Control Theory Applications*, 141(4), 209-215.

Fahlman, S.E. (1989) Fast learning variations on back-propagation: an empirical study. *In: Proceedings of Connectionist Models Summer School*, San Mateo, California.

Fahlman, S.E. and Lebiere, C. (1990) The cascade-correlation learning architecture, *In: Advances in Neural Information Processing Systems 2*, San Mateo, California.

Franzini, J. B., Freyberg, D. L., Linsley, R.K. and Tchobanoglous, G. (1992) *Water Resource Engineering*, McGraw-Hill, New York, NY.

French, M.N., Krajewski, W.F. and Cuykendall, R.R. (1992) Rainfall forecasting in space and time using neural networks, *J. of Hydrology*, 137, 1-31.

Georgakakos, A.P., Georgakakos, K.P. and Baltas, E.A. (1990) A state-space model for hydrologic river routing, *Water Resources Research*, 26(5),827-838.

Hagan, M.T. and Menhaj, M. (1994) Training feedforward networks with the Marquardt algorithm, *IEEE Transactions on Neural Networks*, 5(6):

Hagan, M. T., Demuth, H.B. and Beale, M. (1995) *Neural Network Design*, PWS Publishing Co., Boston, MA.

Haykin, S. (1994) *Foundations of Neural Networks*, Macmillan, New York, NY.

Hwang, J.N., Jou, I.C., Lay, S.R. and You, S.S. (1996) The cascade-correlation learning: a projection pursuit learning perspective, IEEE *Transactions on Neural Networks*, 7(2),278-288.

Karunanithi, N., Grenney, W.J., Whitley, D. and Bovee, K. (1994) Neural networks for river flow prediction, *J. Computing in Civil Engineering, ASCE*, 8(2), 201-219.

Kao, J.J. (1996) Neural network for determining DEM-based model drainage pattern. *J. of Irrigation and Drainage Engineering, ASCE*, 122(2),112-121.

Kraijenhoff, D.A. and Moll, J.R. (1986) *River Flow Modeling and Forecasting*, D. Reidel Publishing Co., Dordrecht, Holland.

Kuschewski, J.G., Hui, S., and Zak, S.H. (1993) Application of feedforward neural networks to dynamical system identification and control, *IEEE Tansactions on Control System Technology*, 1(1), 37-49.

Levin, A.U. and Narendra, K. (1996) Control of nonlinear dynamical systems using neural networks-part II: observability, identification, and control, *IEEE Transactions on Neural Networks*, 7(1), 30-42.

Luo, F.L. and Unbehauen, R. (1998) *Applied Neural Networks for Signal Processing*, Cambridge University Press, New York, NY.

Mason, J.C., Price, R.K. and Tem'me, A. (1996) A neural network model of rainfall-runoff using radial basis functions, *J. Hydraulic Research*, 34(4), 537-548.

McCarthy, G.T. (1938) The unit hydrograph and flood routing. *Conference of North Atlantic Division, USCE*, New London, Connecticut.

Mohan, S. (1995) Parameter estimation of nonlinear Muskingum models using genetic algorithm, *Department of Civil Engineering, Indian Institute of Technology*, Chennai, India.

Montes, S. (1998) *Hydraulics of open-channel flow*, ASCE Press, Reston, VA.

Narendra, K.S. and Parthasarathy, K. (1990) Identification and control of dynamical systems using neural networks, *IEEE Transactions on Neural Networks*, 1(1), 4-27.

Papageorgiou, M. and Messmer, A. (1989). Flow control in a long river reach, *Automatica*, 177-183.

Ranjithan, S., Eheart, J.W. and Garrett, J.H. (Jr) (1993) Neural networks-based screening for groundwater reclamation and uncertainty, *Water Resources Research*, 29(3), 563-574.

Raman, H. and Chandramouli, V. (1996) Deriving a general operating policy for reservoirs using neural networks, *J. Water Resources Planning and Management*, ASCE, 122(5),342-347.

Reddy, J.M. (1997) Optimal regulation of river flows using neural networks, presented at the *Wyoming Water Development Commission Budget Hearing*, Cheyenne, WY

Reddy, J.M. (1995) Neuro-control of irrigation canals, *Proceedings of the First International Water Resources Engineering Conference,* San Antonio, TX

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986) Learning representations by back-propagating errors, *Nature (London)*, 323, 533-536.

United States Bureau of Reclamation, Great Plains Region, web site: www.gp.usbr.gov

Water Resources Data System at the Wyoming Water Resources Center (1996-1998), Laramie, WY.

Warwick, K. (1995) A critique of neural networks for discrete-time linear control, *International J. Control*, 61(6), 1253-1264.

Werbos, P.J. (1990) Back-propagation through time: what it does and how to do it, *revised from IEEE Proc.* 78(10), 1550-1560.

Wilamowski, B.M. and Torvik, L. (1993) Modification of gradient computation in the back-propagation algorithm, Presented at *ANNIE'93 - Artificial Neural Networks in Engineering,* St. Louis, Missouri, November 14-17, 1993; also in Intelligent Engineering Systems Through Artificial Neural Networks vol. 3, pp. 175-180, ed. C. H. Dagli, L. I. Burke, B. R. Fernandez, J. Gosh, R.T., ASME PRESS, New York.

Wilamowski, B. M. (1996) *Neural Networks and Fuzzy Systems*, Chapters 124.1 to 124.8 in The Electronic Handbook, CRC Press.

Zurada, J.M. (1992) *Introduction to Artificial Neural Systems*. West Publishing Company, New York, NY.