ARTIFICIAL NEURAL NETWORKS IN HYDROLOGY. I: PRELIMINARY CONCEPTS

By the ASCE Task Committee on Application of Artificial Neural Networks in Hydrology[1]

ABSTRACT: In this two-part series, the writers investigate the role of artificial neural networks (ANNs) in hydrology. ANNs are gaining popularity, as is evidenced by the increasing number of papers on this topic appearing in hydrology journals, especially over the last decade. In terms of hydrologic applications, this modeling tool is still in its nascent stages. The practicing hydrologic community is just becoming aware of the potential of ANNs as an alternative modeling tool. This paper is intended to serve as an introduction to ANNs for hydrologists. Apart from descriptions of various aspects of ANNs and some guidelines on their usage, this paper offers a brief comparison of the nature of ANNs and other modeling philosophies in hydrology. A discussion on the strengths and limitations of ANNs brings out the similarities they have with other modeling approaches, such as the physical model.

BACKGROUND

This is a first part of a two-part series prepared by the ASCE Task Committee on Application of Artificial Neural Networks in Hydrology. The material in this paper is of a basic nature and is targeted towards hydrologists who are essentially beginners in this field. The development of artificial neural networks (ANNs) began approximately 50 years ago (McCulloch and Pitts 1943), inspired by a desire to understand the human brain and emulate its functioning. Within the last decade, it has experienced a huge resurgence due to the development of more sophisticated algorithms and the emergence of powerful computation tools. Extensive research has been devoted to investigating the potential of artificial neural networks (ANNs) as computational tools that acquire, represent, and compute a mapping from one multivariate input space to another (Wasserman 1989). Mathematically, an ANN is often viewed as a universal approximator. The ability to identify a relationship from given patterns make it possible for ANNs to solve large-scale complex problems such as pattern recognition, nonlinear modeling, classification, association, and control. Although the idea of artificial neural networks was proposed by McCulloch and Pitts (1943) over fifty years ago, the development of ANN techniques has experienced a renaissance only in the last decade due to Hopfield's effort (Hopfield 1982) in iterative autoassociable neural networks. A tremendous growth in the interest of this computational mechanism has occurred since Rumelhart et al. (1986) rediscovered a mathematically rigorous theoretical framework for neural networks, i.e., back-propagation algorithm. Consequently, ANNs have found applications in such diverse areas as neurophysiology, physics, biomedical engineering, electrical engineering, computer science, acoustics, cybernetics, robotics, image processing, financing, and others.

Since the early nineties, ANNs have been successfully used in hydrology-related areas such as rainfall-runoff modeling, stream flow forecasting, ground-water modeling, water quality, water management policy, precipitation forecasting, hydrologic time series, and reservoir operations. The application of ANNs as an alternative modeling tool for certain hydrologic problems is the subject of the second part of the series. The goal of this paper is to give a brief description of artificial neural networks, summarizing the commonly used algorithms and guidelines for applying ANNs to hydrologic problems, describing the similarities and differences between ANNs and other modeling approaches, and discussing their strengths and limitations.

INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS

An ANN is a massively parallel-distributed information-processing system that has certain performance characteristics resembling biological neural networks of the human brain (Haykin 1994). ANNs have been developed as a generalization of mathematical models of human cognition or neural biology. Their development is based on the following rules:

1. Information processing occurs at many single elements called nodes, also referred to as units, cells, or neurons,
2. Signals are passed between nodes through connection links.
3. Each connection link has an associated weight that represents its connection strength.
4. Each node typically applies a nonlinear transformation called an activation function to its net input to determine its output signal.

A neural network is characterized by its architecture that represents the pattern of connection between nodes, its method of determining the connection weights, and the activation function (Fausett 1994). Caudill presented a comprehensive description of neural networks in a series of papers (Caudill, 1987, 1988, 1989). A typical ANN consists of a number of nodes that are organized according to a particular arrangement. One way of classifying neural networks is by the number of layers: single (Hopfield nets); bilayer (Carpenter/Grossberg adaptive resonance networks); and multilayer (most backpropagation networks). ANNs can also be categorized based on the direction of information flow and processing. In a feed-forward network, the nodes are generally arranged in layers, starting from a first input layer and ending at the final output layer. There can be several hidden layers, with each layer having one or more nodes. Information passes from the input to the output side. The nodes in one layer are connected to those in the next, but not to those in the same layer. Thus, the output of a node in a layer is only a dependent on the inputs it receives from previous layers and the corresponding weights. On the other hand, in a recurrent ANN, information flows through the nodes in both directions, from the input to the output side and vice versa. This is generally achieved by recycling previous network outputs as current inputs, thus allow-

[1]Rao S. Govindaraju, Assoc. Prof., Purdue Univ., School of Civ. Engrg., 1284 Civ. Engrg. Build., West Lafayette, IN 47907-1284.

Note. Discussion open until September 1, 2000. Separate discussions should be submitted for the individual papers in this symposium. To extend the closing date one month, a written request must be filed with the ASCE Manager of Journals. The manuscript for this paper was submitted for review and possible publication on August 20, 1998. This paper is part of the ***Journal of Hydrologic Engineering***, Vol. 5, No. 2, April, 2000. ©ASCE, ISSN 1084-0699/00/0002-0115–0123/$8.00 + $.50 per page. Paper No. 19090.

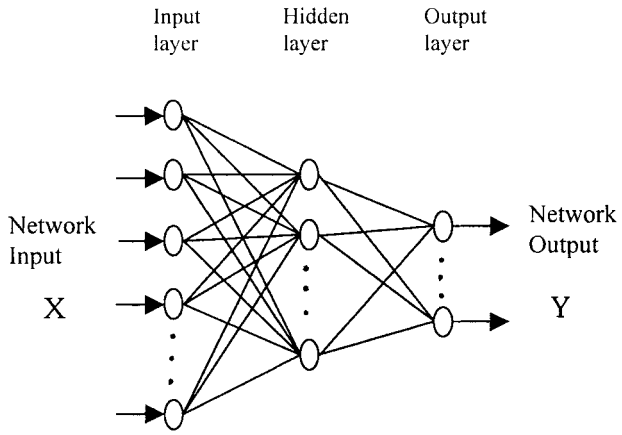

FIG. 1. Configuration of Feedforward Three-Layer ANN

ing for feedback. Sometimes, lateral connections are used where nodes within a layer are also connected. This paper will focus on feedforward and recurrent networks, since they are commonly used in hydrologic problems.

In most networks, the input (first) layer receives the input variables for the problem at hand. This consists of all quantities that can influence the output. The input layer is thus transparent and is a means of providing information to the network. The last or output layer consists of values predicted by the network and thus represents model output. The number of hidden layers and the number of nodes in each hidden layer are usually determined by a trial-and-error procedure. The nodes within neighboring layers of the network are fully connected by links. A synaptic weight is assigned to each link to represent the relative connection strength of two nodes at both ends in predicting the input-output relationship. Fig. 1 shows the configuration of a feedforward three-layer ANN. These kinds of ANNs can be used in a wide variety of problems, such as storing and recalling data, classifying patterns, performing general mapping from input pattern (space) to output pattern (space), grouping similar patterns, or finding solutions to constrained optimization problems. In this figure, **X** is a system input vector composed of a number of causal variables that influence system behavior, and **Y** is the system output vector composed of a number of resulting variables that represent the system behavior.

MATHEMATICAL ASPECTS

A schematic diagram of a typical jth node is displayed in Fig. 2. The inputs to such a node may come from system causal variables or outputs of other nodes, depending on the layer that the node is located in. These inputs form an input vector $\mathbf{X} = (x_1, \ldots, x_i, \ldots, x_n)$. The sequence of weights leading to the node form a weight vector $\mathbf{W}_j = (w_{1j}, \ldots, w_{ij}, \ldots, w_{nj})$, where w_{ij} represents the connection weight from the ith node in the preceding layer to this node.

The output of node j, y_j, is obtained by computing the value

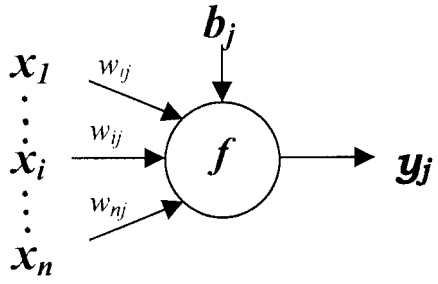

FIG. 2. Schematic Diagram of Node *j*

of function f with respect to the inner product of vector $\mathbf{X}$ and $\mathbf{W}_j$ minus b_j, where b_j is the threshold value, also called the bias, associated with this node. In ANN parlance, the bias b_j of the node must be exceeded before it can be activated. The following equation defines the operation:

$$y_j = f(\mathbf{X} \cdot \mathbf{W}_j - b_j) \tag{1}$$

The function f is called an activation function. Its functional form determines the response of a node to the total input signal it receives. The most commonly used form of $f(\cdot)$ in (1) is the sigmoid function, given as

$$f(t) = \frac{1}{1 + e^{-t}} \tag{2}$$

The sigmoid function is a bounded, monotonic, nondecreasing function that provides a graded, nonlinear response. This function enables a network to map any nonlinear process. The popularity of the sigmoid function is partially attributed to the simplicity of its derivative that will be used during the training process. Some researchers also employ the bipolar sigmoid and hyperbolic tangent as activation functions—both of which are transformed from the sigmoid function. A number of such nodes are organized to form an artificial neural network.

NETWORK TRAINING

In order for an ANN to generate an output vector $\mathbf{Y} = (y_1, y_2, \ldots, y_p)$ that is as close as possible to the target vector $\mathbf{T} = (t_1, t_2, \ldots, t_p)$, a training process, also called learning, is employed to find optimal weight matrices $\mathbf{W}$ and bias vectors $\mathbf{V}$, that minimize a predetermined error function that usually has the form:

$$E = \sum_P \sum_p (y_i - t_i)^2 \tag{3}$$

Here, t_i is a component of the desired output $\mathbf{T}$; y_i = corresponding ANN output; p = number of output nodes; and P = number of training patterns. Training is a process by which the connection weights of an ANN are adapted through a continuous process of stimulation by the environment in which the network is embedded. There are primarily two types of training—supervised and unsupervised. A supervised training algorithm requires an external teacher to guide the training process. This typically implies that a large number of examples (or patterns) of inputs and outputs are required for training. The inputs are cause variables of a system and the outputs are the effect variables. This training procedure involves the iterative adjustment and optimization of connection weights and threshold values for each of nodes. The primary goal of training is to minimize the error function by searching for a set of connection strengths and threshold values that cause the ANN to produce outputs that are equal or close to targets. After training has been accomplished, it is hoped the ANN is then capable of generating reasonable results given new inputs. In contrast, an unsupervised training algorithm does not involve a teacher. During training, only an input data set is provided to the ANN that automatically adapts its connection weights to cluster those input patterns into classes with similar properties. There are occasions when a combination of these two training strategies leads to reinforcement learning. A score or grade is used to rate the network performance over a series of training patterns. Most hydrologic applications have utilized supervised training. The manner in which the nodes of an ANN are structured is closely related to the algorithm used to train it. The rest of this section includes several commonly used training algorithms.

Back-Propagation

Back-propagation is perhaps the most popular algorithm for training ANNs [see, for instance, Wasserman (1989) and Fausett (1994) for a detailed description]. It is essentially a gradient descent technique that minimizes the network error function [(3)]. Each input pattern of the training data set is passed through the network from the input layer to the output layer. The network output is compared with the desired target output, and an error is computed based on (3). This error is propagated backward through the network to each node, and correspondingly the connection weights are adjusted based on equation

$$\Delta w_{ij}(n) = -\varepsilon^* \frac{\partial E}{\partial w_{ij}} + \alpha^* \Delta w_{ij}(n-1) \tag{4}$$

where $\Delta w_{ij}(n)$ and $\Delta w_{ij}(n-1)$ = weight increments between node i and j during the nth and $(n-1)$th pass, or epoch. A similar equation is written for correction of bias values. In (4), ε and α are called learning rate and momentum, respectively. The momentum factor can speed up training in very flat regions of the error surface and help prevent oscillations in the weights. A learning rate is used to increase the chance of avoiding the training process being trapped in a local minima instead of global minima. The back-propagation algorithm involves two steps. The first step is a forward pass, in which the effect of the input is passed forward through the network to reach the output layer. After the error is computed, a second step starts backward through the network. The errors at the output layer are propagated back toward the input layer with the weights being modified according to (4). Back-propagation is a first-order method based on the steepest gradient descent, with the direction vector being set equal to the negative of the gradient vector. Consequently, the solution often follows a zigzag path while trying to reach a minimum error position, which may slow down the training process. It is also possible for the training process to be trapped in the local minimum despite the use of a learning rate. This algorithm is described in detail in Appendix I. In the next section, we describe the conjugate gradient method that can help alleviate this problem.

Conjugate Gradient Algorithms

The conjugate gradient method was first applied to general unconstrained optimization problems by Fletcher and Reeves (1964). Unlike back-propagation, the conjugate gradient method does not proceed along the direction of the error gradient, but in a direction orthogonal to the one in the previous step. This prevents future steps from influencing the minimization achieved during the current step. Fletcher and Reeves (1964) has shown that any minimization method developed by the conjugate gradient algorithm is quadratically convergent. If it is used in the case of the nonquadratic iteration problem, such as error equation (3), a convergence criterion is required. If $P(n)$ is used to denote the direction vector at the nth iteration of back-propagation, (4) can be rewritten as

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \varepsilon \mathbf{P}(n) \tag{5}$$

where $W(n+1)$ and $W(n)$ = weight vectors of some node at the $(n+1)$th and nth iteration; and ε = learning rate (Haykin 1994). The initial direction vector is set equal to the negative gradient vector $\mathbf{g}(n)$ at the initial point $n = 0$, i.e., $\mathbf{P}(0) = -\mathbf{g}(0)$. Each successive direction vector is computed by a linear combination of the current gradient vector and the previous direction vector as

$$\mathbf{P}(n+1) = -\mathbf{g}(n+1) + \beta(n)\mathbf{P}(n) \tag{6}$$

Here, $\beta(n)$ is a time-dependent parameter, which Fletcher and Reeves (1964) defined as

$$\beta(n) = \frac{\mathbf{g}^T(n+1)\mathbf{g}(n+1)}{\mathbf{g}^T(n)\mathbf{g}(n)} \tag{7}$$

The conjugate gradient algorithm can speed up the training process in many cases.

Radial Basis Function

A radial basis function (RBF) network can be considered as a three-layer network in which the hidden layer performs a fixed nonlinear transformation with no adjustable parameters (Leonard et al. 1992). This layer consists of a number of nodes and a parameter vector called a ''center,'' which can be considered the weight vector of the hidden layer. The standard Euclidean distance is used to measure how far an input vector is from the center. For each node, the Euclidean distance between the center and the input vector of the network input is computed and transformed by a nonlinear function that determines the output of the nodes in the hidden layer. The output layer then combines these results in a linear fashion. The output y of an RBF network is computed by the equation

$$y = f(u) = \sum_{i=1}^{n} w_i R_i(\mathbf{x}) + w_0 \tag{8}$$

where w_i = connection weight between the hidden neuron and output neuron; w_0 = bias; and $\mathbf{x}$ = input vector. The functions $R_i : R^n \Rightarrow R$ are radial basis functions that have the general form

$$R_i(\mathbf{x}) = \varphi\|\mathbf{x} - \mathbf{c}_i\| \tag{9}$$

The $\varphi(\cdot)$ has a maximum value at the origin and decays rapidly as its argument tends to infinity. The function of $\varphi(\cdot)$ is required to approach zero as the Euclidean distance increases between an input vector and the center. While there are various choices for $\varphi(\cdot)$, a general class of radial basis functions is described by the Gaussian function

$$R_i = -\exp\left(-\sum_{i=1}^{n} \frac{\|\mathbf{x}_i - \mathbf{c}_i\|^2}{2\sigma_{ij}^2}\right) \tag{10}$$

where $\mathbf{c}_i^T = [c_{i1}, c_{i2}, \ldots, c_{in}]$ is the center of the receptive field; and σ_{ij} = width of the Gaussian function. The major task of RBF network design is to determine center $\mathbf{c}$. The simplest and easiest way may be to choose the centers randomly from the training set. The second approach is to use the k-means technique of clustering input training set into groups and choose the center of each group as the center. Also, $\mathbf{c}$ can be treated as a network parameter along with w_i and adjusted through error-correction training. After the center is determined, the connection weights w_i between the hidden layer and output layer can be determined simply through ordinary back-propagation training.

The primary difference between the RBF network and back-propagation is in the nature of the nonlinearities associated with hidden nodes. The nonlinearity in back-propagation is implemented by a fixed function such as a sigmoid. The RBF method, on the other hand, bases its nonlinearities on the data in the training set. Once all the basis functions in the hidden layer have been found, the network only needs to learn at the output layer in a linear summation fashion.

Cascade Correlation Algorithm

Fahlman and Lebiere (1990) developed the cascade correlation algorithm. It differs from other approaches in that it starts with a minimal network without any hidden nodes and grows during the training by adding new hidden units one by one, maximizing the impact of the new node on the network error, creating a multilayer structure. Once a new hidden node has been added to the network, its input-side weights are fro-

zen. The hidden nodes are trained in order to maximize the correlation between output of the nodes and output error. A training cycle is divided into two phases. First, the output nodes are trained to minimize the total output error. Then a new node is inserted and connected to every output note and all previous hidden nodes. The new node is trained to correlate with the output error. The addition of new hidden nodes is continued until maximum correlation between the hidden nodes and error is attained. The cascade correlation algorithm is unique in that the architecture is determined as a part of the training process. The steps for this algorithm may be summarized as follows (adapted from Thirumaliah and Deo 1998):

1. Start with inputs and output nodes only.
2. Train the network over the training data set by the delta rule.
3. Add a new hidden node. Connect it to all input nodes as well as to other existing hidden nodes. Training of this node is based on maximization of overall correlation S between its output and network error:

$$S = \sum_o \left| \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E}_o) \right| \tag{11}$$

where V_p = output of the new hidden node for pattern p; V = average output over all patterns; $E_{p,o}$ = network output error for output node o on pattern p; and E_o = average network error over all patterns. Pass the training data set one by one and adjust input weights of the new node after each training set until S does not change appreciably.
4. Once training of the new node is done, that node is installed as a hidden node of the network. The input side weights are frozen, and the output side weights are trained again.
5. Go to step 3, and repeat the procedure until the network attains a prespecified minimum error within a fixed number of training epochs.

Recurrent ANNs

We briefly describe recurrent networks herein without going into the details of the algorithm. Such networks typically use a variant of back-propagation for training. The spatial-temporal variability of many hydrologic processes requires the estimation procedure to be dynamic. Such a dynamic relationship can be modeled by carefully chosen artificial neural networks. In the simplest case, a node computes the weighted sum of its inputs passed through a nonlinear activation function. While feedforward neural networks are more popular, they lack the feedback connections necessary to provide a dynamic mode. There are, however, more recent explorations of neural networks that have feedback connections and are thus inherently dynamic in nature.

Essentially, there are three ways that ''memory'' can be introduced into static neural networks. These are (in increasing order of complexity and capability):

1. Tapped delay line models: In these models, the network has past inputs explicitly available (through a tapped delay line) to determine its response at a given point in time (Mozer 1989). Thus, the temporal pattern is converted to a spatial pattern which can then be learned through, say, classic back-propagation.
2. Context models or partial recurrent models: These models retain the past output of nodes instead of retaining the past raw inputs. For example, the output of the hidden layer neurons of a feedforward network can be used as inputs to the network along with the true inputs (Elman 1990; Kothari and Agyepong 1997). These ''network derived'' inputs are also called context inputs. When the interconnections carrying the context inputs are fixed, classical back-propagation can be used for training the network. More complex variations of this basic idea include self-feedback in the context inputs or deriving the context inputs from other locations in the network (Hertz and Palmer 1991).
3. Fully recurrent models: These models employ full feedback and interconnections between all nodes. Algorithms to train fully recurrent models are significantly more complex in terms of time and storage requirements (Pineda 1987, 1989; Rohwer and Forrest 1987; Almeida 1988).

Context models fall somewhere between the simplicity of a tapped delay line model and the complexity of a fully recurrent network. In many situations they provide competitive solutions.

One such neural network algorithm that can be applied to a network of arbitrary connectivity is recurrent back-propagation (Pineda 1987, 1989; Rohwer and Forrest 1987; Almeida 1988). Islam and Kothari (2000) provide a brief overview of recurrent back-propagation including its mathematical aspects and implementation details.

Self-Organizing Feature Maps

Unlike feed-forward and recurrent neural networks that are primarily used for approximation and classification, self-organizing feature maps (SOFMs) are typically used for density estimation or for projecting patterns from high-dimensional to low-dimensional space. This projection is nonparametric and is obtained by mapping input patterns into the responses of nodes arranged in a lattice (two-dimensional or sometimes three-dimensional). SOFM, originally proposed by Kohonen (1989, 1990), has found widespread use as a method of mapping for high-dimensional data. Compared with current approaches, SOFM-based representation of spatial data (e.g., hydraulic conductivity, precipitation, topography, etc.) offers two distinct advantages: (1) nonparametric estimation of the underlying distribution; and (2) a representation that fully preserves the topological structure of the underlying distribution. More specifically, SOFMs have a competitive layer of nodes, arranged in a lattice, with each node connected to all the inputs through adjustable weights. At the beginning of the training process, these weights are randomly initialized. When an input pattern (say, soil moisture measurement at a location) is presented as an input, the first step in the training of a SOFM is to compute a matching value for each node in the competitive layer. This value measures the extent to which the weights of each node match the corresponding values of the input pattern. The node that has the closest match to the input is identified as a winning unit. Interconnection weights of the winning unit and its nearest neighbors (in the lattice) are updated to more closely match the input pattern. Next, another input pattern is chosen from the data set and the process is repeated until the interconnection weights stop changing. The result of this training is a topological map in which the asymptotic local density of the weights approaches that of the training data. Similarities among patterns are mapped into similar weights of neighboring neurons. Islam and Kothari (2000) provide a brief overview of SOFM, including its mathematical aspects and implementation details, with an example of characterizing remotely sensed soil moisture data over a large area.

ANNs AND OTHER MODELING APPROACHES IN HYDROLOGY

Hydrology is the scientific study of water and its properties, distribution, and effects on the earth's surface, soil, and at-

mosphere (McCuen 1997). Most hydrologic processes are highly nonlinear and exhibit a high degree of spatial and temporal variability. They are further complicated by uncertainty in parameter estimates. Hydrologists are often confronted with problems of prediction and estimation of quantities such as runoff, precipitation, contaminant concentrations, and water stages. This kind of information is required in hydrologic and hydraulic engineering design as well as water resources management.

Currently used models in hydrology can be broadly grouped under three categories: empirical, geomorphology based, and physically based. Empirical models treat hydrologic systems (such as a watershed) as a black-box and try to find a relationship between historical inputs (rainfall, temperature, etc.) and outputs (watershed runoff measured at a stream gauge). Lumped catchment models fall under this category (Blackie and Eeles 1985). These methods need long historical records and have no physical basis and, as such, are not applicable for ungauged watersheds. An improvement over these kinds of models are the geomorphology-based models (e.g., Gupta and Waymire 1983; Corradini and Singh 1985). These models represent the watershed structure and the stream network well, but various assumptions concerning the linearity of response of individual watershed units (streams and overland sections) need to be made.

Physically-based models represent the ''physics'' as they are best understood (Freeze and Harlan 1969). Typically, they involve solution of a system of partial differential equations that represent our best understanding of the flow processes within the watershed. For most problems, a numerical solution is sought by discretizing time-space dimensions into a discrete set of nodes. This implies that such models work best when data on the physical characteristics of the watershed are available at the model grid scale. This kind of data is rarely available, even in heavily instrumented research watersheds. The problem of spatial variability and accurate representation of the watershed has led to stochastic watershed models being proposed to account for this uncertainty. Such models have been used in hypothetical studies to demonstrate that the response of watershed units is quite nonlinear. These models suffer from problems such as identification, estimability, and uniqueness of parameter estimation. Even with current computing capabilities, physical representation of the watershed in the model is, at best, an approximation. Physically-based models are applicable to ungauged watersheds in theory, but, almost invariably, they require historical data for model calibration purposes. Despite these limitations, these models have proved to be very useful for many hydrologic problems when utilized appropriately. Engineers and hydrologists continue to model and predict the complex behavior of watershed systems using physically-based models successfully. Michaud and Sorooshian (1994) present a review of previous work comparing the performance of simple versus physically-based models for predicting watershed runoff. It is clear that this problem is still unresolved. It is perhaps for this reason that alternative modeling approaches are still being sought.

Based on this brief discussion, ANNS would have to be classified as empirical models. We call this approach a ''model'' as it has many features in common with other modeling approaches in hydrology. The process of model selection can be considered equivalent to the determination of appropriate network architecture. Similarly, model calibration and validation can be identified with network training, cross training, and testing. In many respects, ANNs are similar to regression-based models in hydrology, except that they do not require specification of a mathematical form. In addition, ANNs are more versatile because of the freedom available with the choice of number of hidden layers and the nodes associated with each of these layers. The ANN structure allows for information to be processed along multiple paths simultaneously, thereby offering opportunities for parallel implementation.

IMPORTANT ASPECTS OF ANN MODELING

There are no fixed rules for developing an ANN, even though a general framework can be followed based on previous successful applications in engineering. Some issues that typically arise while developing an ANN are briefly described in this section.

Selection of Input and Output Variables

The goal of an ANN is to generalize a relationship of the form

$$\mathbf{Y}^m = f(\mathbf{X}^n) \tag{12}$$

where $\mathbf{X}^n$ is an n-dimensional input vector consisting of variables $x_1, \ldots, x_i, \ldots, x_n$; and $\mathbf{Y}^m$ is an m-dimensional output vector consisting of resulting variables of interest $y_1, \ldots, y_i, \ldots, y_m$. We use the term ''generalize'' to imply that the functional form of $f(\cdot)$ in (12) will not be revealed explicitly, but will be represented by the network parameters. In hydrology, the values of x_i can be causal variables such as rainfall, temperature, previous flows, water levels, spatial locations, evaporation, basin area, elevation, slopes, pump operating status, contaminant loads, meteorological data, and so on. The values of y_i can be hydrological responses such as runoff, streamflow, ordinates of a hydrograph, optimal pumping patterns, rain fields, hydraulic conductivities, contaminant concentrations, and others.

The selection of an appropriate input vector that will allow an ANN to successfully map to the desired output vector is not a trivial task. Unlike physically-based models, the set of variables that influence the system are not known a priori. In this sense of nonlinear process identification, an ANN should not be considered as a mere black box. A firm understanding of the hydrologic system under consideration is an important prerequisite for successful application of ANNs. For instance, physical insight into the problem being studied can lead to better choice of input variables for proper mapping. This will help in avoiding loss of information that may result if key input variables are omitted, and also prevent inclusion of spurious inputs that tend to confuse the training process. A sensitivity analysis can be used to determine the relative importance of a variable (Maier and Dandy 1996) when sufficient data is available. The input variables that do not have a significant effect on the performance of an ANN can be trimmed from the input vector, resulting in a more compact network.

Collecting and Preprocessing Data

Most hydrologic data are obtained either from gauges that are placed on site or through remote sensing instruments. Also, either an existing model or laboratory experiments can be used to generate the data patterns for specific applications (French et al. 1992; Ranjithan et al. 1993; Rogers and Dowla 1994; Smith and Eli 1995; Minns and Hall 1996; and others). Again, there appears to be no fixed method for determining the number of input-output data pairs that will be required. To ensure a good approximation, Carpenter and Barthelemy (1994) stated that the number of data pairs used for training should be equal to or greater than the number of parameters (weights) in the network. An optimal data set should be representative of the probable occurrence of an input vector and should facilitate the mapping of the underlying nonlinear process. Inclusion of unnecessary patterns could slow network learning.

In contrast, an insufficient data set could lead to poor learning. This makes it useful to analyze and preprocess the data before it is used for an ANN application. Routine procedures such as plotting and examining the statistics are sometimes effective in judging the reliability of the data and possibly to remove outliers. In many cases, the data needs to be encoded, normalized, or transformed before being applied to an ANN.

Designing an ANN

This important step involves the determination of the ANN architecture and selection of a training algorithm. An optimal architecture may be considered the one yielding the best performance in terms of error minimization, while retaining a simple and compact structure. No unified theory exists for determination of such an optimal ANN architecture. Often, more than one ANN can generate similar results. The numbers of input and output nodes are problem dependent. They are equal to n and m in (12). The flexibility lies in selecting the number of hidden layers and in assigning the number of nodes to each of these layers. A trial-and-error procedure is generally applied to decide on the optimal architecture. As discussed earlier, the cascade-correlation-training algorithm is an efficient method to find the optimal architecture.

The potential of feed-forward neural networks can be attributed to three main factors (Kothari and Agyepong 1996): (1) multilayered feedforward neural networks do not need an explicit mathematical equation relating inputs and outputs; (2) a feed-forward network with a single hidden layer with an arbitrary number of sigmoidal hidden nodes can approximate any continuous function; and (3) a feedforward network with a single hidden layer of m sigmoidal nodes achieves an integrated squared error of $O(1/m)$ while a linear combination of a set of m fixed basis functions achieves an integrated squared error of $O(1/m^{2/d})$, where d is the dimension of the input (Barron 1993). Points 1 and 3 above refer to computational superiority of feedforward ANN, while 2 hints to an existence theorem that establishes the capabilities of a feedforward ANN. It does not, however, allow for a systematic determination of the number of hidden nodes to use in a given situation. The number of hidden layer neurons significantly influence the performance of a network; with too few nodes the network will approximate poorly, while with too many nodes it will overfit the training data.

The influence of the size of a neural network on its generalization performance is well known (Baum and Haussler 1989; Agyepong and Kothari 1997). Bishop (1995) provides an excellent review of proposed approaches that allow the determination of network architecture with acceptable performance on the training and generalization data. Some of the popular techniques include network growing (e.g., Gallant 1986; Nadal 1989; Fahlman and Lebiere 1991; Cios and Liu 1992; Bose and Garga 1993; Kwok and Yeung 1995) and network pruning (e.g., Mozer and Smolensky 1989; Karnin 1990; LeCun et al. 1990; Hassibi and Stork 1993; Reed 1993). These algorithms treat the network structure as an optimization parameter along with the weights. Pruning algorithms generally start with a large network and proceed by removing weights to which sensitivity of the error is minimal. Growing methods, on the other hand, typically start with a small network and add nodes with full connectivity to nodes in the existing network when a suitably chosen measure (e.g., entropy, covariance, etc.) stops decreasing. An alternative to these methods is called soft weight sharing (Nowlan and Hinton 1992), where groups of weights are encouraged to have equal values, allowing for a reduction in the effective number of free parameters in the network. Soft weight sharing can train a large network with a small amount of training data (Agyepong and Kothari 1997); however, to ensure convergence to good solutions, proper initialization of the weights is necessary.

Agyepong and Kothari (1997) have shown that the use of lateral connections in a feedforward network leads to a controlled assignment of role in the hidden layer neurons beginning with the fringe nodes, leaving nodes in the center of the hidden layer unsupervised. Consequently, their network behaves like a growing algorithm without the explicit need to add hidden units and like soft weight sharing due to functionally similar neurons in the center of the hidden layer. This localization of the weight sharing algorithm of Agyepong and Kothari (1997) allows a systematic determination of the number of hidden layer neurons needed for a given learning task.

Training and Cross Training

This is similar to the idea of calibration that is an integral part of most hydrologic modeling studies. The available data set is generally partitioned into three parts for training, cross training, and validation. The purpose of training is to determine the set of connection weights and nodal thresholds that cause the ANN to estimate outputs that are sufficiently close to target values. The dataset reserved for training is used to achieve this goal. This fraction of the complete data to be employed for training should contain sufficient patterns so that the network can mimic the underlying relationship between input and output variables adequately. The weights and threshold values are assigned small random values initially (usually, $-0.3 \sim 0.3$). During training, these are adjusted based on the error, or the difference between ANN output and the target responses. This adjustment can be continued recursively until a weight space is found, which results in the smallest overall prediction error. However, there is the danger of overtraining a network in this fashion, also referred as overfitting. This happens when the network parameters are too fine-tuned to the training data set. It is as if the network, in the process of trying to ''learn'' the underlying rule, has started trying to fit the noise component of the data as well. In other words, overtraining results in a network that memorizes the individual examples, rather than trends in the data set as a whole. When this happens, the network performs very well over the data set used for training, but shows poor predictive capabilities when supplied with data other than the training patterns. To prevent this kind of overfitting, a cross training procedure is usually recommended. The goal of this procedure is to stop training when the network begins to overtrain. The second portion of the data is reserved for this purpose. After the adjustment of network parameters with each epoch, the network is used to find the error for this data set. Initially, errors for both the training and cross training data sets go down. After an optimal amount of training has been achieved, the errors for the training set continue to decrease, but those associated with the cross training data set begin to rise. This is an indication that further training will likely result in the network overfitting the training data. The process of training is stopped at this time, and the current set of weights and thresholds are assumed to be the optimal values. The network is ready for use as a predictive tool. If the available data set is too small for partitioning, the simplest way to prevent overtraining is to stop training when the mean square error ceases to decrease significantly.

Model Validation

Similar to other modeling approaches in hydrology, the performance of a trained ANN can be fairly evaluated by subjecting it to new patterns that it has not seen during training. The performance of the network can be determined by computing the percentage error between predicted and desired values. In addition, plotting the model output versus desired re-

sponse can also be used to assess ANN performance. Since finding optimal network parameters is essentially a minimization process, it is advisable to repeat the training and validation processes several times to ensure that satisfactory results have been obtained.

Some Other Issues

Some ANN applications (see Part II of this two-part paper) have stressed the importance of scaling the input/output quantities before presenting them to the network. For problems exhibiting high nonlinearity, the variables are scaled to fall between a range of 0 to 1, or some other suitable range. This kind of scaling tends to smooth the solution space and averages out some of the noise effects. However, there is some danger of losing information through this procedure.

Initialization of weights and threshold values is an important consideration. The closer the initial guess is to the optimum weight space, the faster the training process. However, there is no way of making a good initial guess of the weights, and they are initialized in a random fashion. Usually, small random weights are suggested.

STRENGTHS AND LIMITATIONS

The following are some of the reasons ANNs have become an attractive computational tool:

1. They are able to recognize the relation between the input and output variables without explicit physical consideration.
2. They work well even when the training sets contain noise and measurement errors.
3. They are able to adapt to solutions over time to compensate for changing circumstances.
4. They possess other inherent information-processing characteristics and once trained are easy to use.

Very often, in hydrology, the problems are not clearly understood or are too ill-defined for a meaningful analysis using physically-based methods. Even when such models are available, they have to rely on assumptions that make ANNs seem more attractive. Moreover, ANNs routinely model the nonlinearity of the underlying process without having to solve complex partial differential equations. Unlike regression-based techniques, there is no need to make assumptions about the mathematical form of the relationship between input and output. The presence of noise in the inputs and outputs is handled by an ANN without severe loss of accuracy because of distributed processing within the network. This, along with the nonlinear nature of the activation function, truly enhances the generalizing capabilities of ANNs and makes them desirable for a large class of problems in hydrology.

Although several studies indicate that ANNs have proven to be potentially useful tools in hydrology (see Part II), their disadvantages should not be ignored. The success of an ANN application depends both on the quality and the quantity of data available. This requirement cannot be easily met, as many hydrologic records do not go back far enough. Quite often, the requisite data is not available and has to be generated by other means, such as another well-tested model. Even when long historic records are available, we are not certain that conditions remained homogeneous over this time span. Therefore, data sets recorded over a system that is relatively stable and unaffected by human activities are desirable. Representing temporal variations is often achieved by including past inputs/outputs as current inputs. However, it is not immediately clear how far back one must go in the past to include temporal effects. This makes the resulting ANN structure more complicated. Yet another major limitation of ANNs is in the lack of physical concepts and relations. This has been one of the primary reasons for the skeptical attitude towards this methodology. The fact that there is no standardized way of selecting network architecture also receives criticism. The choice of network architecture, training algorithm, and definition of error are usually determined by the user's past experience and preference, rather than the physical aspects of the problem.

CONCLUSION

This paper serves as an introduction to artificial neural networks (ANNS) with emphasis on their application to hydrologic problems. It presents a brief description of ANNs, the underlying concept and mathematical aspects, and the role of ANNs relative to other modeling approaches in hydrology. Some popular ANN architectures and algorithms are discussed. Guidelines for application of ANNs to problems in hydrology are presented. The merits and shortcomings of this methodology are discussed. The second paper takes a closer look at what impact ANNs have had in several areas of hydrology. Towards the end of that paper, the current lacunae in ANN applications are revisited, with some suggestions for future research in this area.

APPENDIX I. BACK-PROPAGATION ALGORITHM

In a multilayer feed-forward neural network, connections between nodes of different layers exist, and no such connections exist between the nodes within the same layer. The inputs are presented to a network at the input layer, and the stimulation is passed through the network from input side to output side. Such a network, with learning governed by a generalized delta rule, is typically called a back-propagation neural network. This algorithm was originally developed by Werbos (1974) in his Ph.D. dissertation at Harvard University. Nevertheless, its powerfulness was not recognized and appreciated for many years. Rumelhart et al. (1986) rediscovered the algorithm and made it popular by demonstrating how to train the hidden neurons for a complex mapping problems. Their work played a crucial role in the resurrection of the whole neural network field. The network consists of an input layer, an output layer, and a number of hidden layers. At each node in a layer the information is received, stored, processed, and communicated further to nodes in the next layer. All the weights are initialized to small random numeric values at the beginning of training. These weights are updated or modified iteratively using the generalized delta rule or the steepest-gradient descent principle. The training process is stopped when no appreciable change is observed in the values associated with the connection links or some termination criterion is satisfied. Thus, the training of a backpropagation network consists of two phases: a forward pass, during which the processing of information occurs from the input layer to the output layer; and a backward pass, when the error from the output layer is propagated back to the input layer and the interconnections are modified. The algorithm, is given by Fausett (1994), is as follows:

- Step 0. Initialize weights. (Set to small random values.)
- Step 1. While stopping condition is false, do Steps 2–9.
- Step 2. For each training pair of set, do Steps 3–8.

Feed-forward:

- Step 3. Each input unit (X_i, $i = 1, 2, \ldots, n$) receives input signal x_i and sends this signal to all units in the next layer (the hidden units).

- Step 4. Each hidden unit (Z_j, $j = 1, 2, \ldots, p$) sums its weighted input signals.

$$Zin_j = v_{oj} + \sum x_i v_{ij} \quad \text{for} \quad i = 1, 2, \ldots, n \tag{13}$$

where v_{ij} = connection weight and v_{oj} = bias value, and applies its activation function to compute its output signal:

$$Z_j = f(Zin_j) \tag{14}$$

and sends this signal to all units in the following layer (output nodes). Typically, ''f'' is the sigmoidal nonlinear function, defined as

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{15}$$

- Step 5. Each output unit (Y_k, $k = 1, 2, \ldots, m$) sums its weighted input signals

$$Yin_k = w_{0k} + \sum z_j w_{jk} \quad \text{for} \quad j = 1, 2, \ldots, p \tag{16}$$

and applies its activation function to compute its output signal:

$$Y_k = f(Yin_k) \tag{17}$$

Back-propagation of error:

- Step 6. Each output unit (Y_k, $k = 1, 2, \ldots, m$) receives a target pattern corresponding to the input training pattern, computes its error information term

$$\delta_k = (t_k - y_k) f'(Yin_k) \tag{18}$$

calculates its weight correction term (used to update w_{jk} later)

$$\Delta w_{jk} = \partial \delta_k Z_j \tag{19}$$

calculates its bias correction term (used to update w_{0k} later)

$$\Delta w_{0k} = \alpha \delta_k \tag{20}$$

and sends δ_k to nodes in the previous layer.
- Step 7. Each hidden unit (Z_k, $j = 1, 2, \ldots, p$) sums its delta inputs (from units in the next layer)

$$\delta in_j = \sum \delta_k w_{jk} \quad \text{for} \quad k = 1, 2, \ldots, m \tag{21}$$

multiplies by the derivative of its activation function to calculate its error information term

$$\delta_j = \delta in_j f'(Zin_j) \tag{22}$$

calculates its weight correction term (used to update v_{ij} later)

$$\nabla v_{ij} = \partial \delta_j x_i \tag{23}$$

and calculates its bias correction term (used to update v_{oj} later)

$$\Delta v_{oj} = \alpha \delta_j \tag{24}$$

Update weights and biases:

- Step 8. Each output node (Y_k, $k = 1, 2, \ldots, m$) updates its bias and weights ($j = 0, 1, \ldots, p$):

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk} \tag{25}$$

Each hidden node (Z_j, $j = 1, 2, \ldots, p$) updates its bias and weights ($I = 0, 1, \ldots, n$):

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij} \tag{26}$$

- Step 9. Test stopping condition.

In spite of the potential capabilities of back-propagation, it suffers from several drawbacks that can lead to problems during training. The most commonly faced problems are long, ambiguous training process, network paralysis, and local minima.

For complex problems, the direction of error reduction with number of epochs is not very clear and may lead to some confusion regarding the continuation of the training process. The training process can be very long and sometimes futile for a bad initial weight space. The two main reasons behind slow learning are problems of step size and moving target (Fahlman and Lebiere 1990). During the search for an optimal weight space, the error is continuously reduced by reducing the step size until an optimal minimum is reached. For most practical scenarios, this local minimum may yield good results or may be the global minimum itself. However, no fixed guidelines are available for the rate at which this step size should be reduced during network training. Reducing the step size by infinitesimal amounts may not be practically feasible. A large step size may propel the search in a very different region of weight space and may yield a poor solution. The problem of moving target arises as the weights have to continuously adjust their values from one output to another for successive patterns. The change in a weight during one training pass may be nullified in the next pass because of a different training pattern. Thus, the attenuation and dilution of weight changes also slows down the process of training.

The problem of network paralysis arises due to the large adjustment of weights in the initial epochs. When all the nodes produce large outputs, the derivative of the activation function can become very small. Since the error sent back from the output layer in the backward pass (or the weight adjustment) is proportional to the derivative of the activation function, the learning process slows down, and weight adjustment might be insignificant. This problem is usually avoided by reducing the step size or learning rate. The problem of local minima is faced by many traditional optimization methods, and such is the case for neural networks. This problem arises due to downward search in a complex, high-dimensional space full of hills, valleys, saddle points, etc. By changing the learning rate or step size, this problem can be avoided to some degree. Wasserman (1989) proposed a back-propagation network that uses certain statistical methods to find global minima.

ACKNOWLEDGMENTS

The Task Committee on Application of Artificial Neural Networks in Hydrology consisted of the following participants: *Chair*—R. S. Govindaraju, Purdue University. *Vice Chair*—A. R. Rao, Purdue University. *Members*—David Leib, Kansas Water Office; Y. M. Najjar, Kansas State University; H. V. Gupta, University of Arizona; A. Hjelmfelt, University of Missouri. *Mail-In Members*—M. Markus, National Weather Service; A. S. Tokar, National Weather Service; S. Islam, University of Cincinnati; J. D. Salas, Colorado State University; C. Ray, University of Hawaii. This paper was prepared with assistance provided by Bin Zhang, a doctoral student at Purdue University. The studies of R. S. Govindaraju and Bin Zhang were supported by NSF Grant EAR-9524578.

APPENDIX II. REFERENCES

Agyepong, K., and Kothari, R. (1997). ''Controlling hidden layer capacity through lateral connections.'' *Neural Computation*, 9(6), 1381–1402.

Almeida, L. B. (1987). ''A learning rule for asynchronous perceptrons with feedback in a combinatorial environment.'' *Proc., IEEE First Int. Conf. on Neural Networks*, Institute of Electrical and Electronics Engineers, New York, 2, 609–618.

Almeida, L. B. (1988). ''Backpropagation in perceptrons with feedback.'' *Neural computers*, R. Eckmiller and Ch. von der Malsburg, eds., Springer Verlag, Berlin, 199–208.

Barron, A. R. (1993). ''Universal approximation bounds for superposition of a sigmoidal function.'' *IEEE Trans. Information Theory*, 39, 930–945.

Baum, E., and Haussler, D. (1989). ''What sized net gives valid generalization.'' *Neural Information Processing Sys.*, 1, 81–90.
Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press, New York.
Blackie, J. R., and Eeles, W. O. (1985). ''Lumped catchment models.'' *Hydrological forecasting*, M. G. Anderson and T. P. Burt, eds., Wiley, New York, 311–345.
Bose, N. K., and Garga, A. K. (1993). ''Neural network design using Voronoi diagrams.'' *IEEE Trans. on Neural Networks*, 4(5), 778–787.
Carpenter, W. C., and Barthelemy, J. (1994). ''Common misconcepts about neural networks as approximators.'' *J. Comp. in Civ. Engrg.*, ASCE, 8(3), 345–358.
Caudill, M. (1987). ''Neural networks primer I.'' *AI Expert.*
Caudill, M. (1988). ''Neural networks primer II, III, IV and V.'' *AI Expert*.
Caudill, M. (1989). ''Neural networks primer VI, VII and VIII.'' *AI Expert*.
Cios, K. J., and Liu, N. (1992). ''A machine leaning method for generation of a neural network architecture: a continuous ID3 algorithm.'' *IEEE Trans. on Neural Networks*, 3(2), 280–291.
Corradini, C., and Singh, V. P. (1985). ''Effect of spatial variability of effective rainfall on direct runoff by a geomorphological approach.'' *J. Hydrol., Amsterdam*, 81, 27–43.
Elman, J. L. (1990). ''Finding structure in time.'' *Cognitive Sci.*, 14, 179–211.
Fahlman, S. E., and Lebiere, C. (1990). ''The cascade-correlation learning architecture.'' *Advances in neural information processing systems 2*, D. S. Touretzky, ed., Morgan Kaufmann, San Mateo, Calif., 524–532.
Fahlman, S. E., and Lebiere, C. (1991). ''The cascade-correlation learning architecture.'' *CMU Tech. Rep. CMU-CS-90-100*, Carnegie Mellon University, Pittsburgh, Pa.
Fausett, L. (1994). *Fundamentals of neural networks*. Prentice Hall, Englewood Cliffs, N.J.
Fletcher, R., and Reeves, C. M. (1964). ''Function minimization by conjugate gradient.'' *Comp. J.*, 7, 149–154.
Freeze, R. A., and Harlan, R. L. (1969). ''Blueprint for a physically-based digital simulated hydrologic response model.'' *J. Hydrol.*, Amsterdam, 9, 237–258.
French, M. N., Krajewski, W. F., and Cuykendal, R. R. (1992). ''Rainfall forecasting in space and time using a neural network.'' *J. Hydrol.*, Amsterdam, 137, 1–37.
Gallant, S. I. (1986). ''Three constructive algorithms for network learning.'' *Proc., 8th Ann. Conf. of the Cognitive Sci. Soc.*, Cognitive Science Society, Ann Arbor, Mich., 652–660.
Gupta, V. K., and Waymire, E. (1983). ''On the formulation of an analytical approach to hydrologic response and similarity at the basin scale.'' *J. Hydrol.*, Amsterdam, 65, 95–123.
Hassibi, B., and Stork, D. G. (1993). ''Second order derivatives for networks pruning: optimal brain surgeon.'' *Proc., Neural Information Processing Sys. 4*, MIT Press, Cambridge, Mass., 164–171.
Haykin, S. (1994). *Neural networks: a comprehensive foundation*. MacMillan, New York.
Hertz, J., and Palmer, R. G. (1991). *Introduction to the theory of neural computation*. Addison-Wesley, Reading, Mass.
Hopfield, J. J. (1982). ''Neural networks and physical systems with emergent collective computational abilities.'' *Proc., Nat. Academy of Scientists*, 79, 2554–2558.
Islam, S., and Kothari, R. (2000). ''Artificial neural networks in remote sensing of hydrologic processes.'' *J. Hydrologic Engrg.*, ASCE, 5(2), 138–144.
Karnin, E. D. (1990). ''A simple procedure for pruning back propagation trained neural networks.'' *IEEE Trans. Neural Networks*, 1, 239–242.
Kohonen, T. (1989). *Self organization and associative memory*. Springer Verlag, New York.
Kohonen, T. (1990). ''The self organizing map.'' *Proc. IEEE*, 78, 1464–1480.
Kothari, R., and Agyepong, K. (1996). ''On lateral connections in feedforward neural networks.'' *Proc., IEEE Int. Conf. on Neural Networks*, Institute of Electrical and Electronics Engineers, New York, 13–18.
Kothari, R., and Agyepong, K. (1997). ''Induced specialization of context units for temporal pattern recognition and reproduction.'' *Proc., IEEE Neural Networks for signal Processing VII*, J. Principe, L. Gile, N. Morgan, and E. Wilson, eds., Institute of Electrical and Electronics Engineers, New York, 131–140.
Kwok, T., and Yeung, D. (1995). ''Constructive feedforward neural networks for regression problems: a survey.'' *Tech. Rep.*, Hong Kong University of Science & Technology, Hong Kong.
LeCun, Y., Denker, J. S., and Solla, S. A. (1990). ''Optimal brain damage.'' *Proc., Neural Information Processing Sys. 2*, MIT Press, Cambridge, Mass., 598–605.
Leonard J. A., Kramer, M. A., and Ungar, L. H. (1992). ''Using radial basis functions to approximate a function and its error bounds.'' *IEEE Trans. on Neural Networks*, 3(4), 624–627.
Maier, H. R., and Dandy, G. C. (1996). ''The use of artificial neural networks for the prediction of water quality parameters.'' *Water Resour. Res.*, 32(4), 1013–1022.
McCuen, R. H. (1997). *Hydrologic analysis and design*, 2nd Ed., Prentice Hall, Upper Saddle River, N.J.
McCulloch, W. S., and Pitts, W. (1943). ''A logic calculus of the ideas immanent in nervous activity.'' *Bull. of Math. Biophys.*, 5, 115–133.
Michaud, J., and Sorooshian, S. (1994). ''Comparison of simple versus complex distributed runoff models on a midsized semiarid watershed.'' *Water Resour. Res.*, 30(3), 593–605.
Minns, A. W., and Hall, M. J. (1996). ''Artificial neural networks as rainfall-runoff models.'' *Hydrological Sci.*, 41(3), 399–417.
Mozer, M. C., and Smolensky, P. (1989). ''Skeletonization: a technique for trimming the fat from a network via relevance assessment.'' *Advances in neural information processing systems 1*, D. Touretzky, ed., Morgan Kaufmann, San Monteo, Calif., 107–115.
Nadal, J. P. (1989). ''Study of a growth algorithm for neural networks.'' *Int. J. Neural Sys.*, 1, 55–59.
Nowlan, S. J., and Hinton, G. E. (1992). ''Simplifying neural networks by soft weight sharing.'' *Neural Computation*, 4, 473–493.
Pineda, F. J. (1987). ''Generalization of back-propagation to recurrent neural networks.'' *Phys. Rev. Lett.*, 59, 2229–2232.
Pineda, F. J. (1989). ''Recurrent back-propagation and the dynamical approach to adaptive neural computation.'' *Neural Computation*, 1, 161–172.
Ranjithan, S., Eheart, J. W., and Rarret, Jr., J. H. (1993). ''Neural network-screening for groundwater reclamation under uncertainty.'' *Water Resour. Res.*, 29(3), 563–574.
Reed, R. (1993). ''Pruning algorithm—a survey.'' *IEEE Trans. Neural Networks*, 4, 740–747.
Rogers, L. L., and Dowla, F. U. (1994). ''Optimization of groundwater remediation using artificial neural networks with parallel solute transport modeling.'' *Water Resour. Res.*, 30(2), 457–481.
Rohwer, R., and Forrest, B. (1987). ''Training time-dependence in neural networks.'' *Proc., IEEE 1st Int. Conf. on Neural Networks*, Institute of Electrical and Electronics Engineers, New York, 2, 701–708.
Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). ''Learning internal representations by error propagation.'' *Parallel distributed processing*, Vol. 1, MIT Press, Cambridge, Mass., 318–362.
Smith, J., and Eli, R. N. (1995). ''Neural-network models of rainfall-runoff process.'' *J. Water Resour. Plng. and Mgmt.*, ASCE, 121(6), 499–508.
Thirumaliah, K., and Deo, M. C. (1998). ''River stage forecasting using artificial neural networks.'' *J. Hydrologic Engrg.*, ASCE, 3(1), 27–32.
Wasserman, P. D. (1989). *Neural computing: theory and practice*. Van Nostrand Reinhold, New York.
Werbos, P. (1974). ''Beyond regression: new tools for prediction and analysis in the behavioral sciences,'' PhD dissertation, Harvard University, Cambridge, Mass.