6. Data-Based Models

6 Data-Based Models

Data sets contain information, often much more than can be learned from just looking at plots of those data. Models based on observed input and output data help us abstract and gain new information and understanding from these data sets. They can also serve as substitutes for more process-based models in applications where computational speed is critical or where the underlying relationships are poorly understood. Models based on data range from the commonly used regression models to models based on, for example, evolutionary or biological processes. This chapter provides a brief introduction to these types of models. They are among a wide range of data-mining tools designed to identify and abstract information and/or relationships from large sets of data.

1. Introduction

Most optimization and simulation models used for water resources planning and management describe, in mathematical terms, the interactions and processes that take place among the various components of the system. These mechanistically or process-based models usually contain parameters whose values are determined from observed data during model calibration. These types of models are contrasted to what are typically called 'black-box' models, or statistical models. Statistical models do not describe physical processes. They attempt to convert inputs (e.g., rainfall, inflows to a reservoir, pollutants entering a wastewater treatment plant or discharged to a river) to outputs (e.g., runoff, reservoir releases, pollutant concentrations in the effluent of a treatment plant or downstream from a point of discharge to a river) using any mathematical equation or expression that does the job.

Regression equations, such as of the forms

$$\text{output} = a + b\,(\text{input}) \qquad (6.1)$$

$$\text{output} = a + b\,(\text{input})^{c} \qquad (6.2)$$

or

$$\text{output} = a + b_1\,(\text{input}_1)^{c1} + b_2\,(\text{input}_2)^{c2} \ldots \qquad (6.3)$$

are examples of such data-based or statistical models. They depend on data consisting of both observed inputs and observed outputs for the estimation of the values of their parameters (a, b, c etc.) and for further refinement of their structure. They lack an explicit, well-defined representation of the processes involved in the transformation of inputs to outputs.

While these statistical models are better at interpolating within the range of data used to calibrate them, rather than extrapolating outside that range (as illustrated in Figure 6.1), many have proven quite successful in representing complex physical systems within those ranges.

Regression equations are examples of data-based modelling methods. Other examples of such methods are based on evolutionary principles and concepts. These are a class of probabilistic search procedures known as evolutionary algorithms (EAs). Such algorithms include genetic algorithms (GAs), genetic or evolutionary programming (GP or EP) and evolutionary strategy (ES). Each of these methods has many varieties but all use computational methods based on natural evolutionary processes and learning. Perhaps the most robust and hence the most common of these methods are genetic algorithms and all their varieties, and genetic programming. Alternatively, an extension of regression is artificial neural networks (ANN). The development and application of black-box statistical models like GP and ANNs emulate larger, deterministic, process-oriented models. Their use may be advantageous when for some reason a

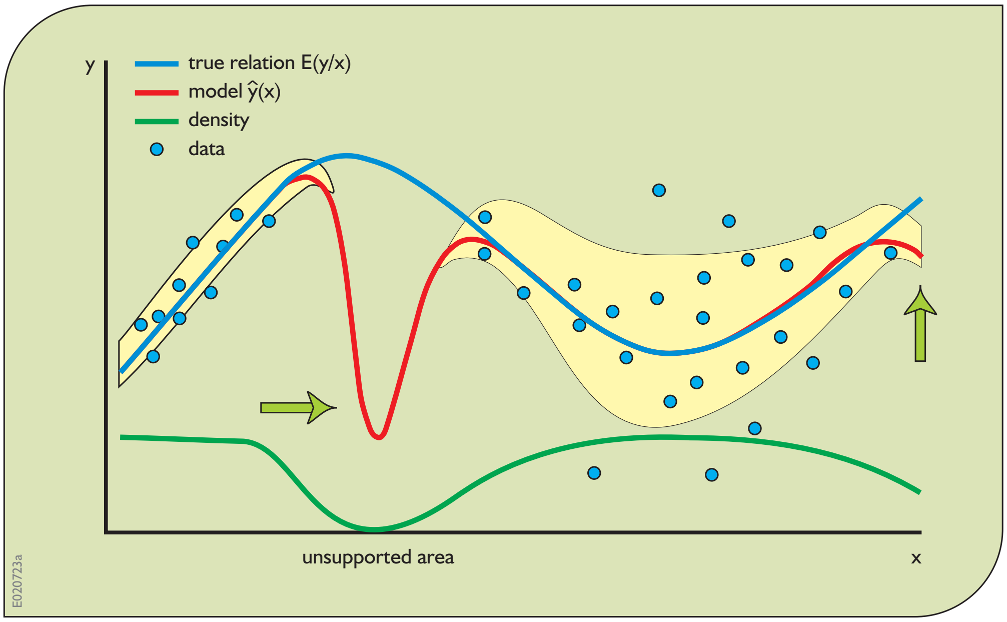

Figure 6.1. Data-driven models are able to estimate relatively accurately within their calibrated ranges, but not outside those ranges. The bottom curve represents the relative density of data used in model calibration. The arrows point to where the model does not predict well.

large number of model solutions must be obtained in a short period of time.

Examples of such situations where multiple solutions of a model must be obtained include sensitivity or uncertainty analysis, scenario evaluations, risk assessment, optimization, inverse modelling to obtain parameter values, and/or when model runs must be extremely fast, as for rapid assessment and decision support systems, real-time predictions/management/control, and so on. Examples of the use of data-driven models for model emulation are given in the following sections.

Genetic algorithms and *genetic programming* are automated, domain-independent methods for evolving solutions to existing models or for producing new models that emulate actual systems, such as rainfall–runoff relationships in a watershed, wastewater removal processes in a treatment plant or discharges of water from a system of natural lakes, each subject to random inputs.

Search methods such as genetic algorithms and genetic programming are inspired by our understanding of biology and natural evolution. They start initially with a number of randomly created values of the unknown variables or a number of black-box models, respectively. The variable values or structure of each of these models is progressively improved over a series of generations. The evolutionary search uses the Darwinian principal of 'survival of the fittest' and is patterned after biological operations including crossover (sexual recombination), mutation, gene duplication and gene deletion.

Artificial neural networks are distributed, adaptive, generally nonlinear networks built from many different processing elements (PEs) (Principe et al., 2000). Each processing element receives inputs from other processing elements and/or from itself. The inputs are scaled by adjustable parameters called weights. The processing elements sum all of these weighted inputs to produce an output that is a nonlinear (static) function of the sum. Learning (calibration) is accomplished by adjusting the weights. The weights are adjusted directly from the training data without any assumptions about the data's statistical distribution or other characteristics (Hagan et al., 1996; Hertz et al., 1991).

The following sections are intended to provide some background helpful to those who may be selecting one among all the available computer codes for implementing a genetic algorithm, genetic program or artificial neural network. These sections are largely based on Babovic and Bojkov (2001) and from material provided by Mynett and Van den Boogaard of Delft Hydraulics.

2. Artificial Neural Networks

2.1. The Approach

Before the development of digital computers, any information processing necessary for thinking and reasoning was carried out in our brains. Much of it still is. Brain-based information processing continues today and will

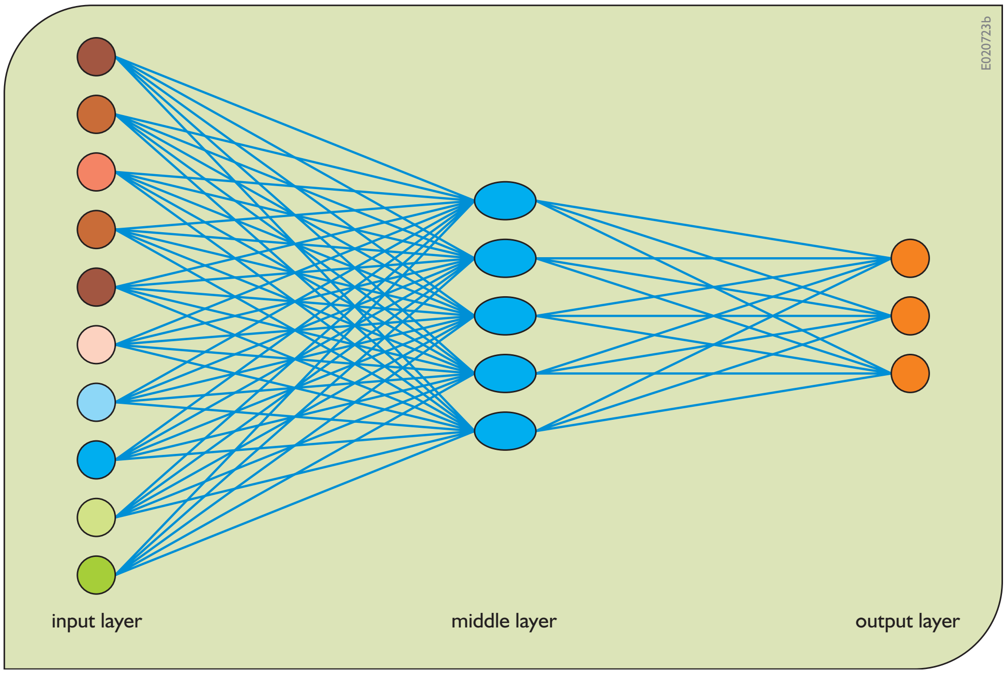

Figure 6.2. A typical multi-layer artificial neural network showing the input layer for ten different inputs, the hidden layer(s), and the output layer having three outputs.

continue in the future even given our continually improving digital processing technology capabilities. While recent developments in information technology (IT) have mastered and outperformed much of the information processing one can do just using brain power, IT has not mastered the reasoning power of our brains. Perhaps because of this, some computer scientists have been working on creating information processing devices that mimic the human brain. This has been termed *neurocomputing*. It operates with networks (ANNs) representing simplified models of the brain. In reality, it is just a more complex type of regression or statistical (black-box) model.

The basic structure of an ANN is shown in Figure 6.2. There are a number of input layer nodes on the left side of the figure and a number of output layer nodes on the right. The middle columns of nodes between these input and output nodes are called *hidden layers*. The number of hidden layers and the number of nodes in each layer are two of the design parameters of any ANN.

Most applications require networks that contain at least three types of layers:

- *The input layer* consists of nodes that receive an input from the external environment. These nodes do not perform any transformations upon the inputs but just send their weighted values to the nodes in the immediately adjacent, usually 'hidden,' layer.
- *The hidden layer(s)* consists of nodes that typically receive the transferred weighted inputs from the input layer or previous hidden layer, perform their transformations on it, and pass the output to the next adjacent layer, which can be another hidden layer or the output layer.
- *The output layer* consists of nodes that receive the hidden-layer output and send it to the user.

The ANN shown in Figure 6.2 has links only between nodes in immediately adjacent layers or columns and is often referred to as a multi-layer perceptron (MLP) network, or a feed-forward (FF) network. Other architectures of ANNs, which include recurrent neural networks (RNN), self-organizing feature maps (SOFMs), Hopfield networks, radial basis function (RBF) networks, support vector machines (SVMs) and the like, are described in more detail in other publications (for example, Haykin, 1999; Hertz et al., 1991).

Essentially, the strength (or weight) of the connection between adjacent nodes is a design parameter of the ANN. The output values O_j that leave a node j on each of its outgoing links are multiplied by a weight, w_j. The input I_k to each node k in each middle and output layer is the sum of each of its weighted inputs, $w_j\, O_j$ from all nodes j providing inputs (linked) to node k.

$$\text{Input value to node } k\text{: } I_k = \sum_j w_j O_j \qquad (6.4)$$

Again, the sum in Equation 6.4 is over all nodes j providing inputs to node k.

At each node k of hidden and output layers, the input I_k is an argument to a linear or non-linear function $f_k(I_k + \theta_k)$, which converts the input I_k to output O_k. The variable θ_k represents a bias or threshold term that influences the horizontal offset of the function. This transformation can take on a variety of forms. A commonly used transformation is a sigmoid or logistic function as defined in Equation 6.5 and graphed in Figure 6.3.

$$O_k = 1/[1 + \exp\{-(I_k + \theta_k)\}] \tag{6.5}$$

The process of converting inputs to outputs at each hidden layer node is illustrated in Figure 6.4. The same process also happens at each output layer node.

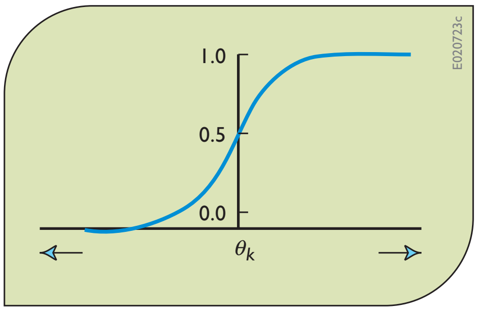

Figure 6.3. The sigmoid or logistic threshold function with threshold θ_k.

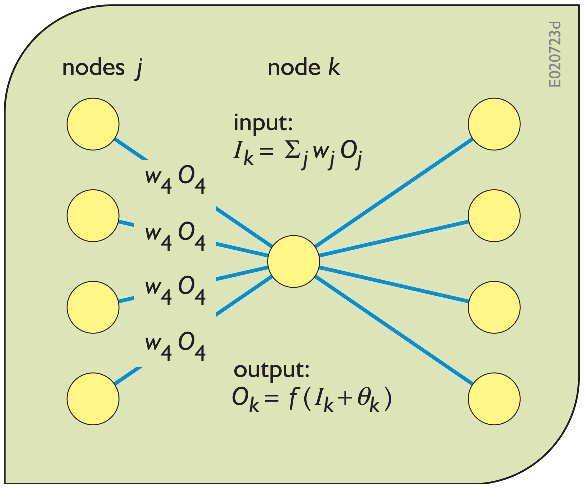

Figure 6.4. A hidden-layer node k converting input values to an output value using a non-linear function f (such as defined by Equation 6.5) in a multi-layer ANN.

The design issues in artificial neural networks are complex and are major concerns of ANN developers. The number of nodes in the input as well as in the output layer is usually predetermined from the problem to be solved. The number of nodes in each hidden layer and the number of hidden layers are calibration parameters that can be varied in experiments focused on getting the best fit of observed and predicted output data-based on the same input data. These design decisions, and most importantly the determination of the values of the weights and thresholds of each connection, are 'learned' during the 'training' of the ANN using predefined (or measured) sets of input and output data.

Some of the present-day ANN packages provide options for building networks. Most provide fixed network layers and nodes. The design of an ANN can have a significant impact on its data-processing capability.

There are two major connection topologies that define how data flows between the input, hidden and output nodes. These main categories are:

- *Feed-forward networks* in which the data flow through the network in one direction from the input layer to the output layer through the hidden layer(s). Each output value is based solely on the current set of inputs. In most networks, the nodes of one layer are fully connected to the nodes in the next layer (as shown in Figure 6.2); however, this is not a requirement of feed-forward networks.
- *Recurrent or feedback networks* in which, as their name suggests, the data flow not only in one direction but in the opposite direction as well for either a limited or a complete part of the network. In recurrent networks, information about past inputs is fed back into and mixed with inputs through recurrent (feedback) connections. The recurrent types of artificial neural networks are used when the answer is based on current data as well as on prior inputs.

Determining the best values of all the weights is called training the ANN. In a so-called supervised learning mode, the actual output of a neural network is compared to the desired output. Weights, which are usually randomly set to begin with, are then adjusted so that the next iteration will produce a closer match between the desired and the actual output. Various learning methods for weight adjustments try to minimize the differences or errors between observed and computed output data. Training consists of presenting input and output data to the network. These data are often referred to as training data. For each input provided to the network, the corresponding desired output set is provided as well.

The training phase can consume a lot of time. It is considered complete when the artificial neural network reaches a user-defined performance level. At this level the network has achieved the desired statistical accuracy as it produces the required outputs for a given sequence of inputs. When no further learning is judged necessary, the resulting weights are typically fixed for the application.

Once a supervised network performs well on the training data, it is important to see what it can do with data it has not seen before. If a system does not give a reasonable output for this test set, this means that the training period should continue. Indeed, this testing is critical to ensure that the network has learned the general patterns involved within an application and has not simply memorized a given set of data.

Smith (1993) suggests the following procedure for preparing and training an ANN:

1. Design a network.
2. Divide the data set into training, validation and testing subsets.
3. Train the network on the training data set.
4. Periodically stop the training and measure the error on the validation data set.
5. Save the weights of the network.
6. Repeat Steps 2, 3 and 4 until the error on the validation data set starts increasing. This is the moment where the overfitting has started.
7. Go back to the weights that produced the lowest error on the validation data set, and use these weights for the trained ANN.
8. Test the trained ANN using the testing data set. If it shows good performance use it. If not, redesign the network and repeat entire procedure from Step 3.

There is a wide selection of available neural network models. The most popular is probably the multi-layer feed-forward network, which is typically trained with static back propagation. They are easy to use, but they train slowly, and require considerable training data. In fact, the best generalization performance is produced if there are at least thirty times more training samples than network weights (Haykin, 1999). Adding local recurrent connections can reduce the required network size, making it less sensitive to noise, but it may get stuck on a solution that is inferior to what can be achieved.

2.2. An Example

To illustrate how an ANN might be developed, consider the simple problem of predicting a downstream pollutant concentration based on an upstream concentration and the streamflow. Twelve measurements of the streamflow quantity, velocity and pollutant concentrations at two sites (an upstream and a downstream site) are available. The travel times between the two measurement sites have been computed and these, plus the pollutant concentrations, are shown in Table 6.1.

Assume at first that the ANN structure consists of two input nodes, one for the upstream concentration and the

travel time (days)	concentration upstream	downstream
2.0	20.0	6.0
2.0	15.0	4.5
1.5	30.0	12.2
1.0	20.0	11.0
0.5	20.0	14.8
1.0	15.0	8.2
0.5	30.0	22.2
1.5	25.0	10.2
1.5	15.0	6.1
2.0	30.0	9.0
1.0	30.0	16.5
0.5	25.0	18.5

E020723e

Table 6.1. Streamflow velocities and pollutant concentrations.

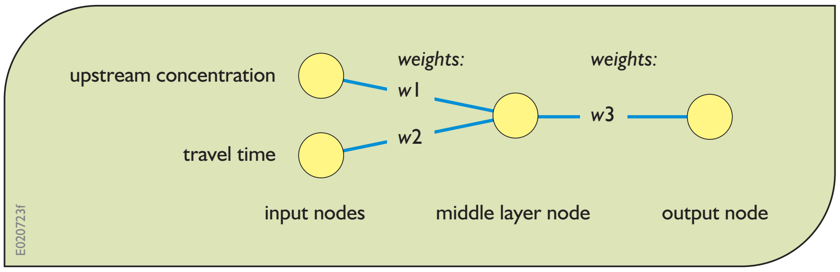

Figure 6.5. Initial ANN for example problem.

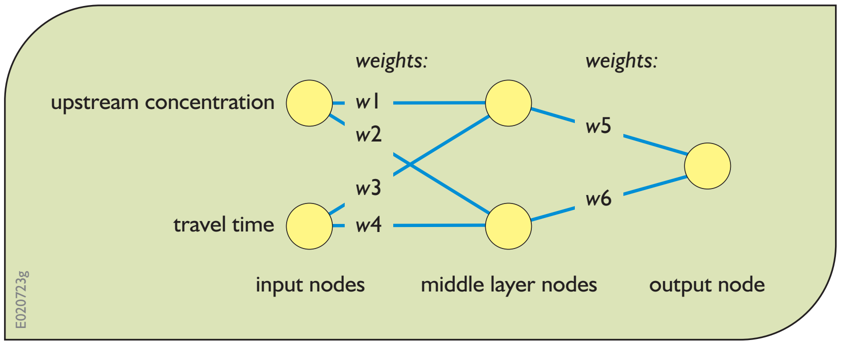

Figure 6.6. Modified ANN for example problem.

Table 6.2. Weights for each link of the ANN shown in Figure 6.6 based on six data sets from Table 6.1. All bias variables (θ_k in Equation 6.5) were 0.

weights	value	weights	value
w_1	0.0	w_5	8.1
w_2	0.0	w_6	-2.8
w_3	-0.6		
w_4	3.9		

other for the travel time, a single hidden layer of only one node, and a single output node, the downstream concentration expressed as a fraction of the upstream concentration. This is shown in Figure 6.5.

The model output is the fraction of the upstream concentration that reaches the downstream site. That fraction can be any value from 0 to 1. Hence the sigmoid function (Equation 6.5) is applied at the middle node and at the output node. Using two or more data sets to train or calibrate this ANN (Figure 6.5) results in a poor fit as measured by the minimum sum of absolute deviations between calculated and measured concentration data. The more data samples used, the worse the fit. This structure is simply too simple. Hence, another node was added to the middle layer. This ANN is shown in Figure 6.6.

Using only half the data (six data sets) for training or calibration, the weights obtained provided a near perfect fit. The weights obtained are shown in Table 6.2.

Next the remaining six data sets were applied to the network with weights set to those values shown in Table 6.2. Again the sum of absolute deviations was essentially 0. Similar results were obtained with increasing numbers of data sets.

The weights in Table 6.2 point out something water quality modellers typically assume, and that is that the fraction of the upstream pollutant concentration that reaches the downstream site is independent of the actual upstream concentration (see Chapter 12). This ANN could have had only one input node, namely that for travel time. This conforms to the typical first order decay function:

Fraction of pollutant concentration downstream per unit concentration upstream $= \exp\{-k^*(\text{travel time})\}$ (6.6)

where the parameter k is the decay rate constant (travel time units^{-1}).

2.3. Recurrent Neural Networks for the Modelling of Dynamic Hydrological Systems

The flexibility, efficiency, speed and emulation capacity of data-based models are particularly relevant within the fields of hydrology, meteorology and hydraulics, where one often has to model dynamic systems and processes that evolve in time. The previous illustration of an ANN was typical of many static ANNs. In training and/or applications the input–output patterns could be processed 'independently' in the sense that an input pattern is not affected by the ANN's response to previous input patterns. The serial order of the input–output patterns does not really matter.

This situation is the same as with the more commonly applied linear or non-linear regression and interpolation techniques where curves or surfaces are fitted to a given set of data points. In fact, standard ANNs are universal function approximators (Cybenko, 1989; Hornik et al., 1989) and, as such, are static non-linear regression models without explicit consideration of time, memory, and time evolution and/or interaction of input–output patterns.

For dynamic systems, the dependencies and/or interaction of input–output patterns cannot be ignored. This is perhaps most recognizable from the state–space property of dynamic models. In a dynamic state–space model, the system's state in time t is based upon the system's state at time $t-1$ (and/or more previous time steps) and external forcings on the system. (See, for example, the applications of dynamic programming to reservoir operation in Chapter 4.)

This state–space property of dynamic models can be included within an ANN to create data-based models better suited for non-linear modelling and the analysis of dynamic systems and time series. This extension leads to the use of recurrent neural networks. These models are equipped with a time propagation mechanism that involves feedback of computed outputs rather than feedback of observed outputs, as shown in Figure 6.7. More detail concerning RNNs can be found in Van den Boogaard et al. (2000).

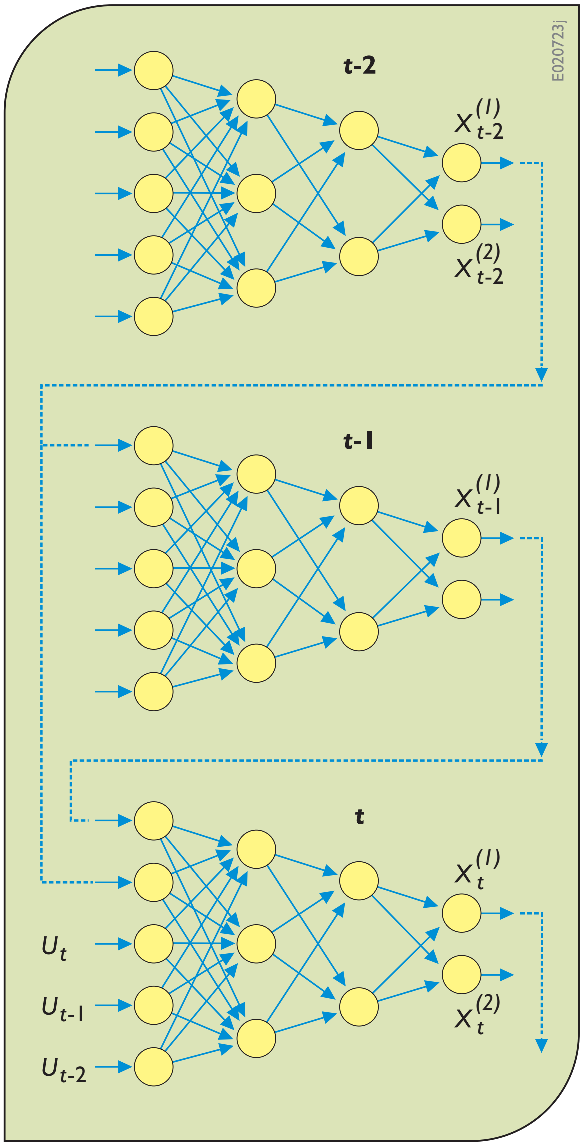

Figure 6.7. Illustration of the architecture of a recurrent neural network (RNN) where the inputs include external data *U* as well as computed data *X* from previous periods *t*.

2.4. Some Applications

Here two hydrological RNN applications are presented. The first case applies a RNN for the emulation of a dynamic conceptual model of a sewerage system. The second case applies a RNN to the modelling of the water balance in Lake IJsselmeer in the Netherlands.

2.4.1. RNN Emulation of a Sewerage System in the Netherlands

A major problem in sewerage systems is overflow during severe rainstorm events. An overflow occurs when the capacity of the sewerage system or treatment plants is exceeded during a rainfall event. Overflow devices are often no more than a storage chamber with a weir as an element for flow control. An overflow results when the upstream water depth exceeds the weir's crest.

Approximately 90% of all sewerage systems in the Netherlands are combined systems, spilling diluted sewage into open water systems during extreme storm events. For most towns, these systems have mild slopes, if any, which implies that the system forms an interconnected network with many loops. When all street sewer lines are included, the resultant networks may have thousands or even tens of thousands of pipes.

For the simulation of such complex sewerage systems, models based on the numerical solution of the Saint Venant equations can be used. One such deterministic model is the SOBEK-URBAN model jointly developed by three organizations in the Netherlands: WL | Delft Hydraulics, RIZA and DHV. As one might expect, this is the principal tool used in the Netherlands for the simulation of sewerage systems (Stelling, 2000). This modelling platform provides an integrated approach for a 1-D simulation of processes in rivers, sewers and drainage systems. Flows in the pipe network and the receiving waters can easily be combined. Another well-known deterministic model applicable for sewerage systems is the INFOWORKS model of Wallingford Software in the UK.

Over the past years, legislation regarding sewage spilling to open water systems has placed more and more constraints on the amounts and frequency of overflows allowed during storm events. To assess whether or not a particular sewerage system meets the current legislation, simulations over a ten-year period are required. In the near future, legislation may require that this period be extended to about twenty-five years.

Simulations of sewerage systems are performed on the basis of recorded historical series of rainstorm events. For a given layout, or for proposed rehabilitation designs (with, say, additional storage capacities), this historical set of storm events must be simulated. For a time period of ten or twenty-five years this will typically involve at least several hundred events where sewer overflows result from storms.

For large systems containing several thousands of pipes, such simulations are, and will continue to be, computationally burdensome despite increased computer speed and memory capacity. Even when restricted to subsets of events with potential overflows, the computational effort will still be large, especially when alternative designs must be compared. For this reason, RNNs as fast model emulators and/or model reduction devices have been developed (Price et al., 1998; Proaño et al., 1998).

The external input consists of a rainfall time series during a storm event. The output of the RNN is a time series of overflow discharges at one or more overflow structures. This output must be zero when no overflow occurs. The RNN is calibrated on the basis of an ensemble of rainfall–overflow combinations with the overflow time series generated by a numerical model. The calibration (and verification) set includes rainstorm events with overflow as well as events without overflows. In this way, the output time series of a few nodes of the numerical model are emulated, rather than the complete state of the sewerage system.

An important question is whether or not the emulation can be based on a limited subset of the large 'original' ensemble of events. Otherwise the input–output of virtually all events must be pre-computed with the numerical model, and the desired reduction of computational cost will not be achieved. Apart from that, emulation would no longer make sense as the frequency and quantity of overflows could then be established directly from the numerical model's predictions.

This model emulation/reduction was applied to the sewerage system of Maartensdijk, a town with about 10,000 inhabitants in the central Netherlands. Based on the system emptying time and the storage/pump capacities, 200 potential overflow events were selected from a rainfall series of ten years. For all forty-four events of the calibration and verification set, a simulation with a SOBEK-URBAN model of the Maartensdijk sewerage system was carried out, providing time series of water depths and discharges at three overflow structures in the system. With these depth and discharge series, a RNN-model was calibrated and verified. The weights of the RNN were optimized for all training events simultaneously and not for each event separately.

This study showed that the water depth rather than the discharge determines the state of the system. In fact, the discharges are a function of the water depths (rating curve), but this relation is not one to one (for example: the discharge is zero for any water depth below the weir's crest). Therefore, discharges cannot be used for the representation of the system state. This aspect nicely illustrates that within black-box procedures too, physical principles or system knowledge must often be used to guarantee a meaningful model set-up.

After several training experiments, a proper architecture of the RNN was identified and accurate emulations of the SOBEK-URBAN model were obtained. Validation tests on events not contained in the training and verification set demonstrated the RNN's capability to emulate the deterministic model.

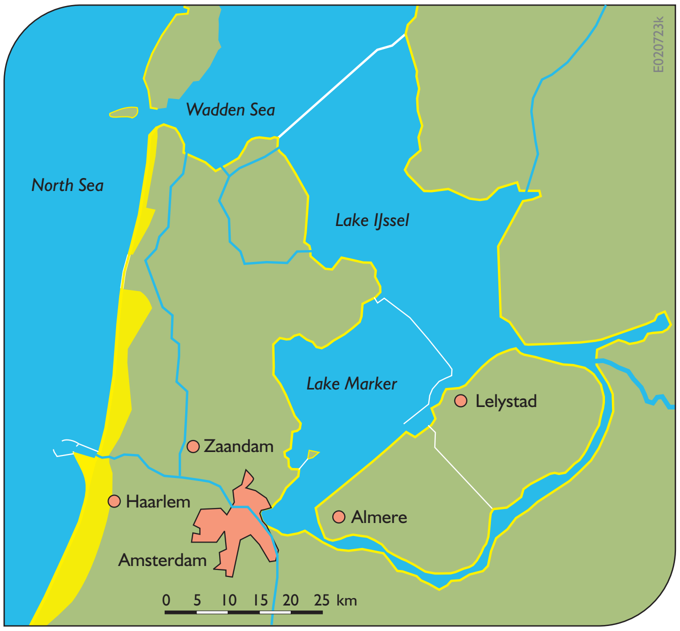

Figure 6.8. Plan view of Lake Ijsselmeer.

2.4.2. Water Balance in Lake IJsselmeer

The most important factors that affect the water balance of Lake IJsselmeer are the inflow of the River IJssel and outflows at the sluices at Den Oever and Kornwerderzand. Figure 6.8 is a map of the area with the positions of the River IJssel and the sluices denoted. The sluices form an interface of the lake to the tidal Dutch Wadden Sea and are used to discharge excess water. This spilling is only possible during low tides when the outside water levels are lower than the water levels within the lake. The spilled volumes are determined by the difference of these water levels. The inner water level can be significantly affected by wind.

An RNN was prepared for the dynamic modelling of the water levels of Lake Ijsselmeer (Van den Boogaard et al., 1998). The model included five external system forcings. These were the discharge of the River IJssel, the north–south and east–west components of the wind, and the outside tidal water levels at the sluices of Kornwerderzand and Den Oever. The (recurrent) output consists of the water level of Lake IJsselmeer. The time step in the model is one day, adopted from the sampling rate of the observed discharges of the River IJssel and the lake's water levels. For synchronization, the time series of the wind and the outer tidal water levels are also represented by daily samples.

The discharges of the River IJssel are a major source of uncertainty. They were not directly observed but on the basis of empirical relations using water-level registrations of the River Rhine at Lobith near the Dutch–German border. These relations may involve errors, especially for winter seasons when water levels and/or discharges are large or even extreme. In all the other external forcings, such as the wind, large uncertainties may also be present.

For applications under operational or real-time conditions (e.g., short to medium-term water-level forecasting) these uncertainties can be taken into account more explicitly by modelling the uncertainties in a statistical sense and applying online data assimilation techniques to improve the model's ability to forecast. Van den Boogaard et al., (2000) present applications of online data assimilation to this situation. The results from this RNN model are probably as good as, if not better than, could be expected from a much more detailed hydraulic process-based model.

For other applications of artificial neural networks applied to problems in meteorology, oceanography, hydraulics, hydrology, and ecology, see Wüst (1995), Minns (1996), Minns and Hall (1996), Scardi (1996), Abrahart et al., (2004), Recknagel et al., (1997), Clair and Ehrman (1998), Hsieh and Tang (1998), Lange (1998), Sanchez et al., (1998), Shen et al., (1998), Van Gent and

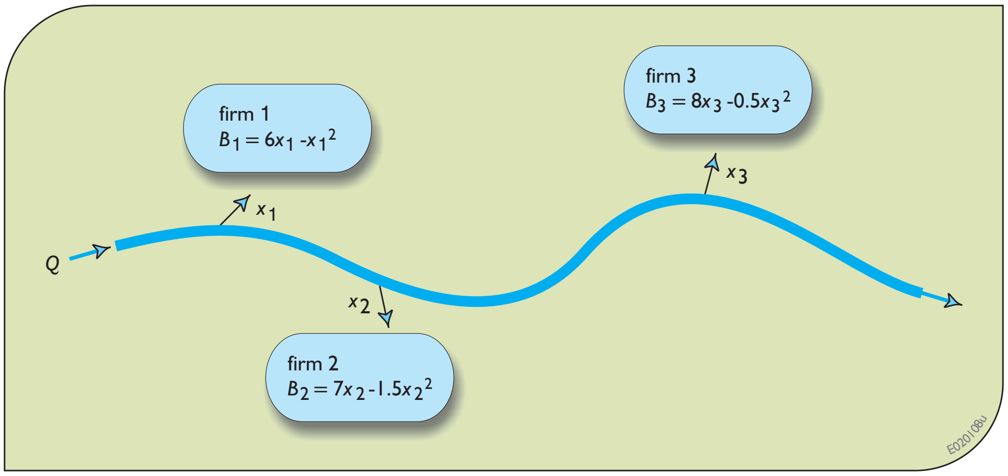

Figure 6.9. Water allocation to three users from a stream having a flow of *Q*.

Van den Boogaard (1998), Wen and Lee (1998) and See and Openshaw (1999). For the identification of other input–output relations see for example Haykin (1999) and Beale and Jackson (1990).

3. Genetic Algorithms

3.1. The Approach

Genetic algorithms are randomized general-purpose search techniques used for finding the best values of the parameters or decision-variables of existing models. It is not a model-building tool like genetic programming or artificial neural networks. Genetic algorithms and their variations are based on the mechanisms of natural selection (Goldberg, 1989). Unlike conventional optimization search approaches based on gradients, genetic algorithms work on a population of possible solutions, attempting to find a solution set that either maximizes or minimizes the value of a function of those solution values. This function is called the objective function. Some populations of solutions may improve the value of the objective function, others may not. The ones that improve its value play a greater role in the generation of new populations of solutions than those that do not.

Each individual solution set contains the values of all the parameters or variables whose best values are being sought. These solutions are expressed as strings of values. For example, if the values of three variables *x*, *y* and *z* are to be obtained, these variables are arranged into a string, *xyz*. If each variable is expressed using three digits, then the string 056004876 would represent $x = 56$, $y = 4$, and $z = 876$. These strings are called *chromosomes*. A chromosome is an array of numbers. The numbers on the chromosome are called *genes*. Pairs of chromosomes from two parents join together and produce offspring, who in turn inherit some of the genes of the parents. Altered genes may result in improved values of the objective function. These genes will tend to survive from generation to generation, while those that are inferior will tend to die.

Chromosomes are usually represented by strings of binary numbers. While much of the literature on genetic algorithms focuses on the use of binary numbers, numbers of any base may be used.

To illustrate the main features of genetic algorithms, consider the problem of finding the best allocations of water to the three water-consuming firms shown in Figure 6.9. Only integer solutions are to be considered. The maximum allocation to any single user cannot exceed 5, and the sum of all allocations cannot exceed the value of Q, say 6.

$$0 \le x_i \le 5 \quad \text{for } i = 1, 2, \text{and } 3. \tag{6.7}$$

$$x_1 + x_2 + x_3 \le 6 \tag{6.8}$$

The objective is to find the values of each allocation that maximizes the total benefits, $B(\boldsymbol{X})$.

$$\text{Maximize } B(\boldsymbol{X}) = \left(6x_1 - x_1^2\right) + \left(7x_2 - 1.5x_2^2\right) + \left(8x_3 - 0.5x_3^2\right) \tag{6.9}$$

A population of possible feasible solutions is generated randomly. A GA parameter is the size of the sample solution population – the number of solutions being considered. The best values of genetic algorithm parameters are usually determined by trial and error.

Using numbers to the base 10, a sample individual solution (chromosome) could be 312, representing the allocations $x_1 = 3$, $x_2 = 1$, and $x_3 = 2$. Another individual solution, picked at random, might be 101. These two individuals or chromosomes, each containing three genes, can pair up and have two children.

The genes of the children are determined by crossover and mutation operations. These pairing, crossover and mutation operations are random. The crossover and mutation probabilities are among the parameters of the genetic algorithm.

Suppose a crossover is to be performed on the pair of strings, 312 and 101. Crossover involves splitting the two solution strings into two parts, each string at the same place. Assume the location of the split was randomly determined to be after the first digit,

3 | 1 2

1 | 0 1

Crossover usually involves switching one part of one string with the corresponding part of the other string. After a crossover, the two new individuals are 301 and 112.

Another crossover approach is to determine for each corresponding pair of genes whether or not they will be exchanged. This would be based on some pre-set probability. For example, suppose the probability of a crossover were set at 0.30. Thus, an exchange of each corresponding pair of genes in a string or chromosome has a 30% chance of being exchanged. The result of this 'uniform' crossover involving, say, only the middle gene in the pair of strings 312 and 101 could be, say, 302 and 111. The literature on genetic algorithms describes many crossover methods for both binary as well as base-10 numbers. The interesting aspect of GA approaches is that they can be, and are, modified in many ways to suit the analyst in the search for the best solution set.

Random mutation operations apply to each gene in each string. Mutation involves changing the value of the gene being mutated. If these strings contained binary numbers, a 1 would be changed to 0, and a 0 would be changed to 1. If numbers to the base 10 are used as they are here, mutation has to be defined. Any reasonable mutation scheme can be defined. For example, suppose the mutation of a base-10 number reduces it by 1, unless the resulting number is infeasible. Hence in this example, a mutation could be defined such that if the current value of the gene being mutated (reduced) is 0, then the new number is 5. Suppose the middle digit 1 of the second new individual, 112, is randomly selected for mutation. Thus, its value changes from 1 to 0. The new string is 102. Mutation could just as well increase any number by 1 or by any other integer value. The probability of a mutation is usually much smaller than that of a crossover.

Suppose these paring, crossover and mutation operations have been carried out on numerous parent strings representing possible feasible solutions. The result is a new population of individuals (children). Each child's fitness, or objective value, can be determined. Assuming the objective function (or fitness function) is to be maximized, the higher the value the better. Adding up all the objective values associated with each child in the population, and then dividing each child's objective value by this total sum yields a fraction for each child. That fraction is the probability of that child being selected for the new population of possible solutions. The higher the objective value, the higher the probability of its becoming a parent in a new population.

In this example the objective is to maximize the total benefit derived from the allocation of water, Equation 6.9. Referring to Equation 6.9, the string 301 has a total benefit of 16.5. The string 102 has a total benefit of 19.0. The sum of these two individual benefits is 35.5. Thus the string 301 has a probability of $16.5/35.5 = 0.47$ of being selected for the new population, and the string 102 has a probability of $19/35.5 = 0.53$ of being selected. Drawing from a uniform distribution of numbers ranging from 0 to 1, if a random number is in the range 0 to 0.47, then the string 301 would be selected. If the random number exceeds 0.47, then the string 102 would be selected. Clearly in a more realistic example the new population size should be much greater than two, and indeed it typically involves hundreds of strings.

This selection or reproduction mechanism tends to transfer to the next generation the better individuals of the current generation. The higher the 'fitness' (i.e. the objective value) of an individual – in other words, the larger the relative contribution to the sum of objective function values of the entire population of individual solutions – the greater will be the chances of that

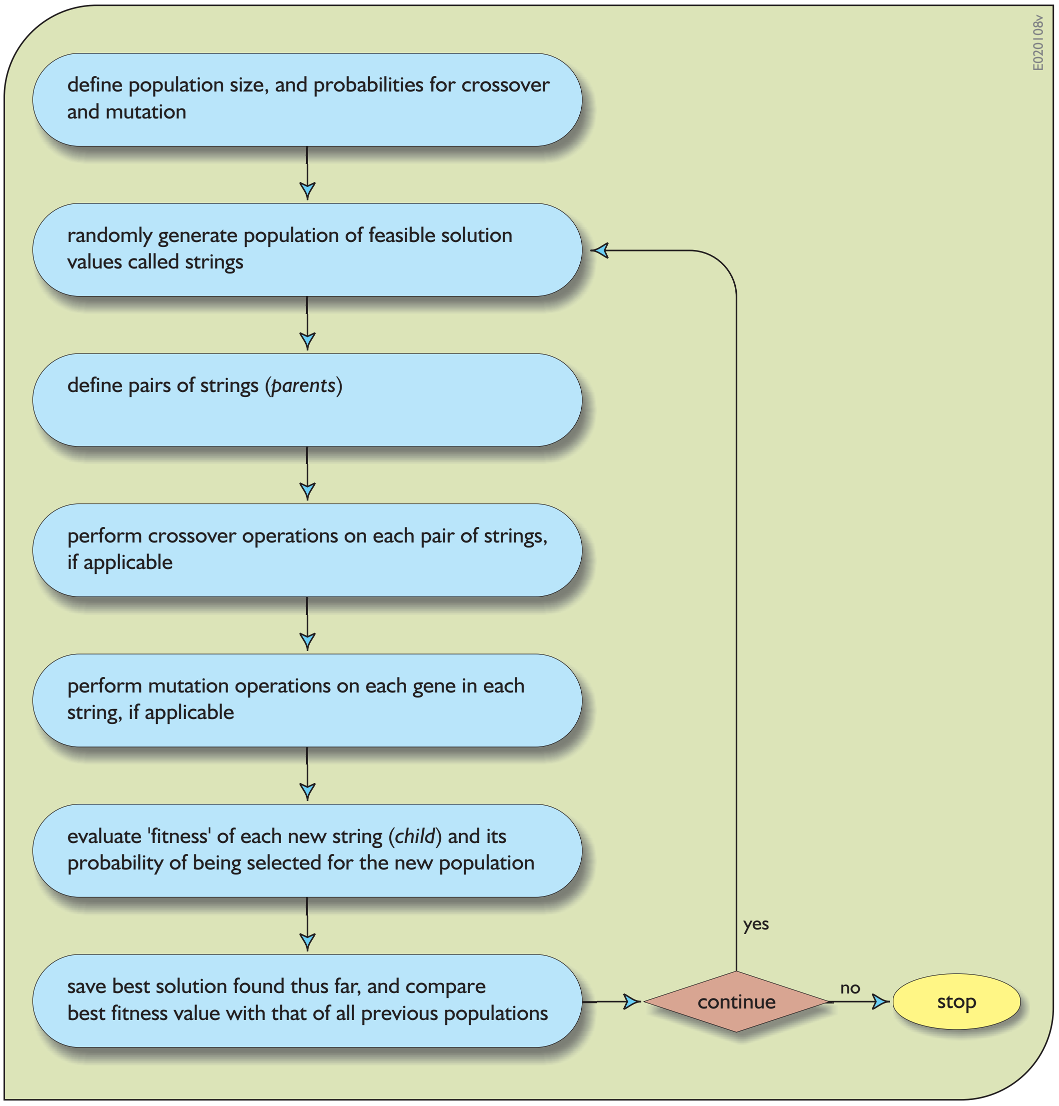
E020108v
define population size, and probabilities for crossover and mutation
randomly generate population of feasible solution values called strings
define pairs of strings (*parents*)
perform crossover operations on each pair of strings, if applicable
perform mutation operations on each gene in each string, if applicable
evaluate 'fitness' of each new string (*child*) and its probability of being selected for the new population
save best solution found thus far, and compare best fitness value with that of all previous populations
continue
yes
no
stop

Figure 6.10. Flow chart of genetic algorithm procedure.

individual string of solution values being selected for the next generation.

Genetic algorithms involve numerous iterations of the operations just described. Each iteration (or generation) produces populations that tend to contain better solutions. The best solution of all populations of solutions should be saved. The genetic algorithm process can end when there is no significant change in the values of the best solution that has been found. In this search process, there is no guarantee this best solution will be the best that could be found, that is, a global optimum.

This general genetic algorithm process just described is illustrated in the flow chart in Figure 6.10.

3.2. Example Iterations

A few iterations with a small population of ten individual solutions for this example water-allocation problem can illustrate the basic processes of genetic algorithms. In practice, the population typically includes hundreds of individuals and the process involves hundreds of iterations. It would also likely include some procedures the modeller/programmer may think would help identify the best solution. Here we will keep the process relatively simple.

The genetic algorithm process begins with the random generation of an initial population of feasible solutions, proceeds with the paring of these solution strings, performs random crossover and mutation operations, computes the probability that each resulting child will be selected for the next population, and then randomly generates the new population. This process repeats itself with the new population and continues until there is no significant improvement in the best solution found from all past iterations.

For this example, we will

1. Randomly generate an initial population of strings of allocation variable values, ensuring that each allocation value (gene) is no less than 0 and no greater than 5. In addition, any set of allocations x_1, x_2 and x_3 that sums to more than 6 will be considered infeasible and discarded.
2. Pair individuals and determine if a crossover is to be performed on each pair, assuming the probability of a crossover is 50%. If a crossover is to occur, we will determine where in the string of numbers it will take place, assuming an equal probability of a crossover between any two numbers.
3. Determine if any number in the resulting individual strings is to be mutated, assuming the probability of mutation of any particular number (gene) in any string (chromosome) of numbers is 0.10. For this example, a mutation reduces the value of the number by 1, or if the original number is 0, mutation changes it to 5. After mutation, all strings of allocation values (the genes in the chromosome) that sum to more than 6 are discarded.
4. Using Equation 6.9, evaluate the 'fitness' (total benefits) associated with the allocations represented by each individual string in the population. Record the best individual string of allocation values from this and previous populations.
5. Return to Step 1 above if the change in the best solution and its objective function value is significant; Otherwise terminate the process.

These steps are performed in Table 6.3 for three iterations using a population of 10.

The best solution found so far is 222: that is, $x_1 = 2$, $x_2 = 2$, $x_3 = 2$. This process can and should continue. Once the process has converged on the best solution it can find, it may be prudent to repeat the process, but this time, change the probabilities of crossover or mutation or let mutation be an increase in the value of a number rather than a decrease. It is easy to modify the procedures used by genetic algorithms in an attempt to derive the best solution in an efficient manner.

4. Genetic Programming

One of the challenges in computer science is to learn how to program computers to perform a task without telling them how to do it. In other words, how can we enable computers to learn to program themselves for solving particular problems? Since the 1950s, computer scientists have tried, with varying degrees of success, to give computers the ability to learn. The name for this field of study is 'machine learning' (ML), a phrase used in 1959 by the first person to make a computer perform a serious learning task, Arthur Samuel. Originally, 'machine learning' meant the ability of computers to program themselves. That goal has, for many years, proven very difficult. As a consequence, computer scientists have pursued more modest goals. A good present-day definition of machine learning is given by Mitchell (1997), who identifies machine learning as the study of computer algorithms that improve automatically through experience.

Genetic programming (GP) aspires to do just that: to induce a population of computer programs or models (objects that turn inputs to outputs) that improve automatically as they experience the data on which they are trained (Banzhaf et al., 1998). Genetic programming is one of many machine-learning methods. Within the machine learning community, it is common to use 'genetic programming' as shorthand for any machine learning system that evolves tree structures (Koza et al., 1992).

While there is no GP today that will automatically generate a model to solve any problem, there are some examples where GP has evolved programs that are better than the best programs written by people to solve a number of difficult engineering problems. Some examples of these human-competitive GP achievements can be seen in Koza (1999), as well as in a longer list on the Internet (www.genetic-programming.com/humancompetitive.html). Since Babovic (1996) introduced the GP paradigm in the field of water engineering, a number of researchers have used the technique to analyse a variety of water management problems.

The main distinctive feature of GP is that it conducts its search for a solution to a given problem by changing model structure rather than by finding better values of model parameters or variables. There is no guarantee, however, that the resulting structure (which could be as simple as regression Equations 6.1, 6.2 or 6.3) will give us any insight into the actual workings of the system.

Table 6.3. Several iterations for solving the allocation problem using genetic algorithms.

	population	crossover		mutation	fitness	sel. prob.	cum. prob.	new pop.
first iteration	230	23|0	220	210	13.5	0.09	0.09	230
	220	22|0	230	230	15.5	0.10	0.19	201
	021	021	021	021	15.5	0.10	0.29	211
	201	201	201	201	15.5	0.10	0.39	132
	301	3|01	321	321	(became infeasible)			301
	221	2|21	201	101	12.5	0.08	0.47	132
	301	30|1	301	301	16.5	0.11	0.58	301
	211	21|1	211	211	21.0	0.14	0.72	021
	132	132	132	132*	26.5	0.19	0.19	230
	310	310	310	210	13.5	0.09	1.00	132
	total fitness				150.0			
second iteration	230	230	230	220	16.0	0.08	0.08	221
	201	201	201	201	15.5	0.08	0.16	132
	211	21|1	212	212*	27.5	0.14	0.30	230
	132	13|2	131	121	20.5	0.10	0.40	212
	301	301	301	301	16.5	0.08	0.48	201
	132	132	132	132	26.5	0.14	0.62	132
	301	3|01	001	001	7.5	0.04	0.66	301
	021	0|21	321	221	23.5	0.12	0.78	221
	230	230	230	230	15.5	0.08	0.86	212
	132	132	132	132	26.5	0.14	1.00	001
	total fitness				195.5			
third iteration	221	22|1	222	222*	30.0	0.15	0.15	221
	132	13|2	131	121	20.5	0.10	0.25	222
	230	230	230	230	15.5	0.07	0.32	230
	212	212	212	112	24.5	0.12	0.44	202
	201	20|1	202	202	22.0	0.09	0.53	131
	132	13|2	131	131	20.0	0.11	0.64	222
	301	301	301	201	15.5	0.08	0.72	012
	221	221	221	221	23.5	0.11	0.83	121
	212	2|12	201	201	15.5	0.08	0.91	202
	001	0|01	012	012	19.5	0.09	1.00	121
	total fitness				206.5			

E020820e

The task of genetic programming is to find at the same time both a suitable functional form of a model and the numerical values of its parameters. To implement GP, the user must define some basic building blocks (mathematical operations and variables to be used); the algorithm then tries to build the model using the specified building blocks.

One of the successful applications of GP in automatic model-building is that of symbolic regression. Here GP searches for a mathematical regression expression in symbolic form that best produces the observed output given the associated input. To perform this task GP uses a physical symbol system divided into two sets. The first set, the so-called terminal set, contains the symbols for independent variables as well as parameter constants as appropriate. The content of this set is determined only by the nature of the problem to be solved. All the basic operators used to form a function $f(\bullet)$ are in the second set, called the functional set. For example, the second set can contain the arithmetic operators $(+, -, *, /)$ and perhaps others such as log, sqrt, and sin as well, depending on the perceived degree of complexity of the regression.

As models have a well-defined grammar and are of variable size and shape, the structures undergoing adaptation in genetic programming are often taken from a class of parse trees. A parse tree is a node-link tree whose nodes are procedures, functions, variables and constants. The sub-trees of a node of a parse tree represent the arguments of the procedure or function of that node. Variables and functions requiring no arguments are called terminals. They are the leaves in the parse tree. Functions that take arguments are branches of a parse tree and are called functions. All terminals in one parse tree compose the terminal set, and all functions in that tree are in the functional set.

One example of a parse tree for the expression a + (b/c) can be depicted as shown in Figure 6.11.

In Figure 6.11 the variables a, b and c are leaves in the parse tree and hence belong to the terminal set. The mathematical operations + and / are functions having two arguments and, hence, are members of the functional set.

The advantage of encoding the induced GP model as a parse tree is that mutations can readily be performed by changing a node's arguments or operator function. Crossover can be achieved by replacing one or more nodes from one individual with those from another without introducing illegitimate syntax changes.

To produce new expressions (individuals) GP requires that two 'parent' expressions from the previous generation be divided and recombined into two offspring expressions. An example of this is illustrated in Figure 6.12. Having two parent expressions presented as parse trees, the crossover operation simply exchanges a branch of one parent with a branch of the other.

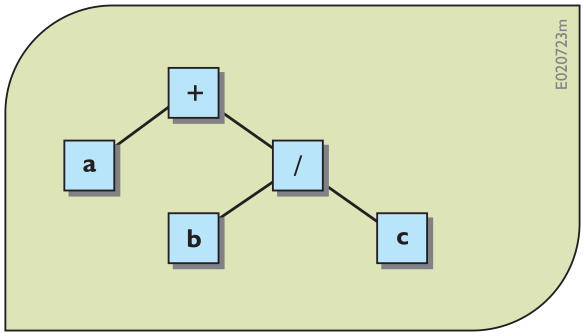

Figure 6.11. A parse tree of the expression a + (b/c).

The result of mutations and crossover operations is the production of two new individuals or children (Figure 6.12 lower). Each child inherits some characteristics from its parents.

This process continues until the fitness of the entire population increases and converges to find the near optimal solution set. The benefit of using the parse tree vehicle in GP can be seen in the crossover operation. This presentation guarantees that every resulting expression is grammatically and semantically correct, regardless of the crossover or mutation site, assuming that it was performed on the correct parents.

Mutation in GP corresponds to a random alteration of the individual parse tree at the branch or node level. There are several possible types of computational mutations. Some examples are:

- *Branch-mutation.* A complete sub-tree is replaced with another, possibly arbitrary sub-tree.
- *Node-mutation.* The value of a single node in the tree is replaced by another, again in an arbitrary way.
- *Constant-mutation.* Similar to node mutation, but here the constant is selected and mutated using white noise.
- *Inversion-mutation.* Inverts the order of operands in an expression in a random way.
- *Edit-mutation.* Edits the parse tree based on semantic equivalence. This mutation does not alter the function produced but just makes it shorter – for example elements like $(x + 0)$ will be replaced everywhere in the expression with x only.

Mutation can affect both the parse tree structure and its information content. With mutation, therefore, the search explores a new domain. It also serves to free the search

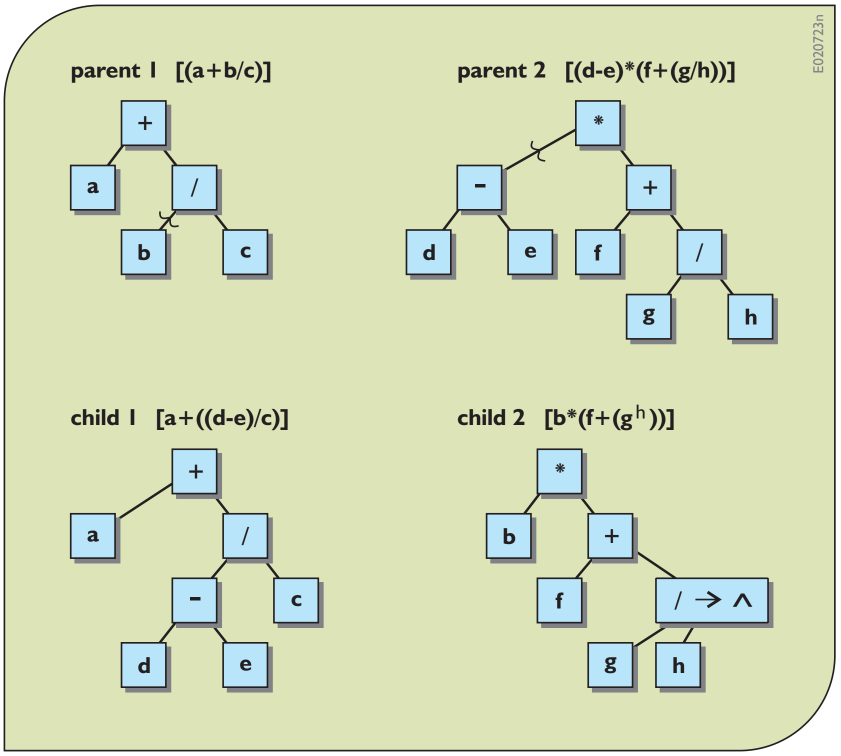

Figure 6.12. An illustration of a crossover operation and mutation (/ → *) operation for genetic programming.

from the possibility of being trapped in local optima. Mutation can be destructive, causing rapid degradation of relatively fit solution sets if the probability of mutation in the algorithm is set too high.

As in most evolutionary algorithms, the models that produce the best 'fit' to the data have the greatest opportunity to become parents and produce children, which is called reproduction. The better models produce the smallest errors, or differences between the calculated output and the observed output.

The basic GP procedure is as follows:

1. Generate the initial population (of size N) of random models (algebraic expressions). Several different initialization procedures exist for GP. Details for these methods can be seen in Babovic and Keijzer (2000).
2. Evaluate each sample model in the population and assign it a fitness value according to how well it solves the problem.
3. Create a new population of size N of models by applying the following two primary operations:
   a) copy the best M ($M < N$) fitted individuals – 'elite' – to the new population (asexual reproduction);
   b) create new L ($L = N - M$) models by genetically recombining randomly chosen parts of two existing models (sexual reproduction).
4. Randomly apply a genetic operator mutation to the offspring and determine the fitness of each model.
5. Repeat steps 3 and 4 until a predetermined stopping criterion is reached.

The stopping criterion in Step 5 of the GP procedure is usually that either (a) a certain number of generations has been produced, or (b) a certain amount of wall-clock time has passed.

Software programs have been written to implement GP. For example GPKernel developed by Babovic and Keijzer (2000) at the Danish Hydraulic Institute (DHI) has been used in applications such as: rainfall–runoff modelling (Babovic and Abbott, 1997; Drecourt, 1999; Liong et al. 2000), sediment transport modelling, salt intrusion in estuaries and roughness estimation for a flow over a vegetation bed (Babovic and Abbott, 1997). More details about GPKernel can be seen in Aguilera (2000).

The challenge in applying genetic programming for model development is not only getting a close fit between observed and predicted outputs, given a set of input data, but also of interpreting the model that is generated to obtain additional understanding of the actual processes taking place. This 'data mining' is discussed in the next section. There are also potential problems in creating a dimensionally correct model if the input data are not dimensionless. As a consequence, many applications using GP seem to require some guidance based on a mix of both physically based and data-based approaches.

5. Data Mining

Data mining is the process of finding new and potentially useful knowledge from data – usually data from large databases. Data mining is also known as *knowledge discovery* (KD). It is the non-trivial extraction of implicit, previously unknown, information from data. It uses machine learning and statistical and visualization techniques to discover and present knowledge in a comprehensible form.

Data mining is a relatively new technology that has the potential to help us learn more from our data. Data mining tools can scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations.

Data mining techniques can be implemented on existing software and hardware platforms to enhance the value of existing information resources, and can be integrated with new products and systems as they are brought online. When implemented on high performance client/server or parallel processing computers, data mining tools can analyse massive databases to seek answers to questions.

5.1. Data Mining Methods

Data mining is supported by three technologies (Babovic, 1998):

- monitoring and data collection
- multiprocessor computers
- data mining algorithms

The core components of data mining technology have been under development for decades in research areas such as statistics, artificial intelligence and machine learning. The current level of development of these techniques, coupled with high-performance relational database engines and broad data-integration efforts, make these technologies practical for current data environments.

Dynamic data access is critical for data navigation applications, and the ability to store, manage and analyse large databases is critical to data mining. Given databases of sufficient size and quality, data mining technology can provide:

- *Automated prediction of trends and behaviours.* Data mining automates the process of finding predictive information in large databases. Questions that traditionally required extensive hands-on analysis can now be answered directly from the data and much more quickly. A typical example of a predictive problem is identifying segments of an ecosystem likely to respond similarly to given events.
- *Automated discovery of previously unknown patterns.* Data mining tools sweep through databases and identify previously hidden patterns. When data mining tools are implemented on high performance parallel processing systems, they can analyse large databases very quickly – often in minutes. Faster processing means that users can automatically experiment with more models to understand complex data. Larger databases, in turn, yield improved predictions.

Some of the techniques commonly used in data mining are:

- *Artificial neural networks*, which are nonlinear predictive models that learn through training and resemble biological neural networks in structure, as discussed earlier.
- *Classification and regression trees* (CARTs), which is used for classification of a dataset. It provides a set of rules that one can apply to a new (unclassified) dataset to predict which records will have a given outcome. It segments a dataset by creating two-way splits.
- *Chi square automatic interaction detection* (CHAID), which is a decision tree technique used for classification of a dataset. It provides a set of rules that one can apply to a new (unclassified) dataset to predict which records will have a given outcome. It segments a dataset by using chi square tests to create multi-way splits. It requires more data preparation than CART.
- *Classification*, which is the process of dividing a dataset into mutually exclusive groups, such that the members of

each group are as 'close' as possible to one another, and different groups are as 'far' as possible from one another, where distance is measured with respect to specific variable(s) one is trying to predict.

- *Clustering*, which is similar to classification, except that distance is measured with respect to all available variables.
- *Data navigation*, which is the process of viewing different dimensions, slices and levels of detail of a multidimensional database. See OLAP, defined below.
- *Data visualization*, which is the visual interpretation of complex relationships in multidimensional data.
- *Decision trees*, which are tree-shaped structures that represent sets of decisions. These decisions generate rules for the classification of a dataset. Specific decision tree methods include classification and regression trees (CARTs) and chi square automatic interaction detection (CHAID).
- *Genetic algorithms and genetic programming*, which search techniques that use processes such as genetic combination, mutation and natural selection in a design based on the concepts of evolution, as discussed earlier.
- *Nearest-neighbour method*, which is a technique that classifies each record in a dataset on the basis of a combination of the classes of the k record(s) most similar to it in a historical dataset. It is sometimes called the k-nearest-neighbour technique.
- *Online analytical processing* (OLAP), which refers to array-oriented database applications that allow users to view, navigate through, manipulate and analyse multidimensional databases.
- *Rule induction*, which is the method of extraction of useful if–then rules from data based on statistical significance.

Many of these technologies have been in use for more than a decade in specialized analysis tools that work with relatively small volumes of data. These capabilities are now evolving to integrate directly with standard databases and online application program platforms.

6. Conclusions

Most computer-based models used for water resources planning and management are physical, mechanistic or process-based models. Builders of such models attempt to incorporate the important physical, biological, chemical, geomorphological, hydrological and other types of interactions among all system components, as appropriate for the problem being addressed and system being modelled. This is done in order to be able to predict possible economic, ecologic, environmental or social impacts that might result from the implementation of some plan or policy. These types of models almost always contain parameters. These need values, and the values of the parameters affect the accuracy of the impact predictions.

This chapter has introduced some data-based methods of modelling. These have included two evolutionary search approaches: genetic algorithms (GA) for estimating the parameter values, and genetic programming for finding models that replicate the real system. In some situations, these biologically motivated search methods, which are independent of the particular model being calibrated, provide the most practical way for model parameter calibrations to be accomplished. These same search methods, sometimes coupled to physically based simulation models, can be used to obtain the values of the unknown decision-variables as well.

While physically based or process-based models are appealing for those who wish to better understand these natural processes, they clearly remain approximations of reality. In some water resources problem applications, the complexities of the real system are considerably greater than the complexities of the models built to simulate them. Hence, it should not be surprising that in some cases statistical or data-based models, which convert input variable values to output variable values in ways that have nothing to do with what happens in reality, may produce more accurate results than physically-based models. This chapter has briefly introduced two such types of data-based model: genetic programming (GP) models and artificial neural networks (ANN). When they work, they often produce results faster than their physical counterparts and as accurately or more so, but only within the range of values observed in the data used to build these models.

Data-driven modelling methods are increasingly being developed and used to gain information from data. They are especially useful when the data sets are large, and where it becomes impractical for any human to sort through it to obtain further insights into the processes that produced the data. Many data-rich problems can be solved by using novel data-driven modelling together with other techniques. Data mining methods are also being increasingly used to gain greater understanding and knowledge from large data sets. Some approaches to data mining are listed in this chapter. Water resources modellers are unlikely to be involved in the development of such data mining techniques, so fortunately, as is the case with GA, GP and ANN methods, many are available on the Internet. Applications of such methods to groundwater modelling, sedimentation processes along coasts and in harbours, rainfall runoff prediction, reservoir operation, data classification, and predicting surge water levels for navigation represent only a small sample of what can be found in the current literature. Some of this literature is cited in the next section.

7. References

ABRAHART, R.J.; KNEALE, P.E. and SEE, L.M. (eds.). 2004. *Neural networks for hydrological modeling*. Leiden, the Netherlands, AA Balkema.

AGUILERA, D.R. 2000. *Genetic Programming with GPKERNEL*. Unpublished, DHI, Copenhagen.

BABOVIC, V. 1996. *Emergence, evolution, intelligence: hydroinformatics*.

BABOVIC, V. 1998. A data mining approach to time series modelling and forecasting. In: V. Babovic and L.C. Larsen (eds.), *Proceedings of the Third International Conference on Hydroinformatics (Hydroinformatics '98)*, Rotterdam, Balkema, pp. 847–56.

BABOVIC, V. and ABBOTT, M.B. 1997. The evolution of equations from hydraulic data. Part II: applications. *Journal of Hydraulic Research*, Vol. 35, No. 3, pp. 411–27.

BABOVIC, V. and BOJKOV, V.H. 2001. *D2K technical report D2K TR 0401-1*. Copenhagen, DHI, April.

BABOVIC, V. and KEIJZER, M. 2000. Genetic programming as a model induction engine. *Journal of Hydroinformatics*, Vol. 2, No. 1, pp. 35–60.

BANZHAF, W.; NORDIN, P.; KELLER, R.E. and FRANNKONE, F.D. 1998. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. San Francisco, Calif., Morgan Kaufmann Publishers, Inc. and dpunkt – Verlag fur Digitale Technologie GmbH.

BEALE, R. and JACKSON, T. 1990. *Neural computing: an introduction*. Bristol, IOP.

CLAIR, T.A. and EHRMAN, J.M. 1998. Using neural networks to assess the influence of changing seasonal climates in modifying discharge, dissolved organic carbon, and nitrogen export in eastern Canadian rivers. *Water Resources Research*, Vol. 34, No. 3, pp. 447–55.

CYBENKO, G. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, Vol. 2, pp. 303–14.

DRECOURT, J.-P. 1999. *Application of neural networks and genetic programming to rainfall–runoff modelling*, D2K Technical Report 0699-1. Copenhagen, DHI.

GOLDBERG, D.E. 1989. *Genetic algorithms in search, optimization, and machine learning*. Reading, Addison-Westley.

HAGAN, M.T.; DEMUTH, H.B. and BEALE, M. 1996. *Neural network design*. Boston, Mass., PWS.

HAYKIN, S. 1999. *Neural networks: a comprehensive foundation*, 2nd edn. Upper Saddle River, N.J., Prentice-Hall.

HERTZ, J.; KROGH, A. and PALMER, R.G. 1991. *Introduction to the theory of neural computation*. Reading, Mass., Addison Wesley.

HORNIK, K.; STINCHCOMBE, M. and WHITE, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, No. 2, pp. 359–66.

HSIEH, W.W. and TANG, B. 1998. Applying neural network models to prediction and data analysis in meteorology and oceanography. *Bulletin of the American Meteorological Society*, Vol. 79, pp. 1855–70.

KOZA, J.R. 1992. *Genetic programming: on the programming of computers by means of natural selection (complex adaptive systems)*. Cambridge, Mass., MIT Press.

KOZA, J.R.; BENNETT, F.H. III; ANDRE, D. and KEANE, M.A. 1999. *Genetic programming III: Darwinian invention and problem solving*. San Francisco, Calif., Morgan Kaufmann.

LANGE, N.T.G. 1998. Advantages of unit hydrograph derivation by neural networks. In: V. Babovic and L.C. Larsen (eds.), *Hydroinformatics '98*, Rotterdam, Balkema, pp. 783–89.

LIONG, S.-Y.; KHU, S.T.; BABOVIC, V.; HAVNOE, K.; CHAN, W.T. and PHOON, K.K. 2000. *Construction of non-linear models in water resources with evolutionary computation*. Singapore, National University of Singapore.

MINNS, A.W. 1996. Extended rainfall–runoff modelling using artificial neural networks. In: A. Müller (ed.), *Hydroinformatics '96*, Rotterdam, Balkema, pp. 207–13.

MINNS, A.W. and HALL, M.J. 1996. Artificial neural networks as rainfall–runoff models. *Hydrological Sciences Journal*, Vol. 41, No. 3, pp. 399–417.

MITCHELL, T.M. 1997. *Machine learning*. New York, McGraw-Hill.

PRICE, R.K.; SAMEDOV, J.N. and SOLOMATINE, D.P. 1998. An artificial neural network model of a generalised channel network. In: V. Babovic and L.C. Larsen (eds.), *Hydroinformatics '98*, Rotterdam, Balkema, pp. 813–18.

PRINCIPE, J.C.; EULIANO, N.R. and LEFEBVRE, W.C. 2000. *Neural and adaptive systems: fundamentals through simulation*. New York, John Wiley.

PROAÑO, C.O.; VERWEY, A.; VAN DEN BOOGAARD, H.F.P. and MINNS, A.W. 1998. Emulation of a sewerage system computational model for the statistical processing of large number of simulations. In: V. Babovic and L.C. Larsen (eds.), *Hydroinformatics '98*, Vol. 2, Rotterdam, Balkema, pp. 1145–52.

RECKNAGEL, F.; FRENCH, M.; HARKONEN P. and YANUNAKA, K.I. 1997. Artificial neural network approach for modelling and prediction of algal blooms. *Ecological Modelling*, No. 96, pp. 11–28.

SANCHEZ, L.; ARROYO, V.; GARCIA, J.; KOEV, K. and REVILLA, J. 1998. Use of neural networks in design of coastal sewage systems. *Journal of Hydraulic Engineering*, Vol. 124, No. 5, pp. 457–64.

SCARDI, M. 1996. Artificial neural networks as empirical models for estimating phytoplankton production. *Marine Ecology Progress Series*, No. 139, pp. 289–99.

SEE, L. and OPENSHAW, S. 1999. Applying soft computing approaches to river level forecasting. *Hydrological Sciences Journal*, Vol. 44, No. 5, pp. 763–78.

SHEN, Y.; SOLOMATINE, D.P. and VAN DEN BOOGAARD, H.F.P. 1998. Improving performance of chlorophyl concentration time series simulation with artificial neural networks. *Annual Journal of Hydraulic Engineering, JSCE*, No. 42, pp. 751–6.

SMITH, M. 1993. *Neural networks for statistical modelling*. New York, Van Nostrand Reinhold.

STELLING, G.S. 2000. A numerical method for inundation simulations. In: Y.N. Yoon, B.H. Jun, B.H. Seoh and G.W. Choi (eds.), *Proceedings of the 4th International Conference on Hydro-Science and Engineering*, Seoul, 26–26 September 2000. Seoul, Korea Water Resources Association.

VAN DEN BOOGAARD, H.F.P.; GAUTAM, D.K. and MYNETT, A.E. 1998. Auto-regressive neural networks for the modelling of time series. In: V. Babovic and L.C. Larsen (eds.), *Hydroinformatics '98*, Rotterdam, Balkema, pp. 741–8.

VAN DEN BOOGAARD, H.F.P.; TEN BRUMMELHUIS, P.G.J. and MYNETT, A.E. 2000. *On-line data assimilation in auto-regressive neural networks*. Proceedings of the Hydroinformatics 2000 Conference, The University of Engineering, Iowa, USA, July 23–27, 2000. Iowa City, University of Iowa.

VAN GENT, M.R.A. and VAN DEN BOOGAARD, H.F.P. 1998. Neural network modelling of forces on vertical structures. In: *Proceedings of the 27th International Conference on Coastal Engineering*, pp. 2096–109. Reston, Va., ASCE press.

WEN, C.-G. and LEE, C.-S. 1998. A neural network approach to multiobjective optimization for water quality management in a river basin. *Water Resources Research*, Vol. 34, No. 3, pp. 427–36.

WÜST, J.C. 1995. Current prediction for shipping guidance. In: B. Kappen and S. Gielen (eds.). *Neural networks: artificial intelligence and industrial applications*. Proceedings of the 3rd Annual SNN Symposium on Neural Networks, Nijmegen, The Netherlands, 14–15 September 1995. London: Springer-Verlag, pp. 366–73.

Additional References (Further Reading)

MCKINNEY, D.C. and LIN, M.D. 1994. Genetic algorithm solution of groundwater management models. *Water Resources Research*, Vol. 30, No. 6, pp. 1897–906.

OLIVERA, R. and LOUCKS, D.P. 1997. Operating rules for multi-reservoir operation. *Water Resources Research*, Vol. 33, No. 4, pp. 839–52.

SOLOMATINE, D.P. and AVILA TORRES, L.A. 1996. Neural network application of a hydrodynamic model in optimizing reservoir operation. In: A. Müller (ed.), *Hydroinformatics '96*, Rotterdam, Balkema, pp. 201–06.