Neuro-Dynamic Programming for the Efficient Management of Reservoir Networks

D. De Rigo[a], A. E. Rizzoli[b], R. Soncini-Sessa[a], E. Weber[a], and P. Zenesi[a]

[a] *Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy*

[b] *IDSIA, Manno, Switzerland (andrea@idsia.ch)*

Abstract: The management of a water reservoir can be improved thanks to the use of stochastic dynamic programming (SDP) to generate management policies which are efficient with respect to the management objectives (flood protection, water supply for irrigation and hydropower generation, respect of minimum environmental flows, etc.). The improvement in efficiency is even more remarkable when the problem involves a reservoir network, that is a set of reservoirs which are interconnected. Unfortunately, SDP is affected by the "curse of dimensionality" and computing time and computer memory occupation can quickly become unbearable. Neuro-dynamic programming (NDP) can sensibly reduce the demands on computer time and memory thanks to the approximation of Bellman functions with Artificial Neural Networks (ANNs). In this paper an application of neuro-dynamic programming to the problem of the management of reservoir networks is presented.

Keywords: Water Reservoir Management; Stochastic Dynamic Programming; Neuro-dynamic Programming

1. INTRODUCTION

The management of water quantity has considerably profited from the advent of computers and the application of Operations Research and Systems Analysis methodologies to solve the problem of finding the optimal release of water in order to satisfy demand for hydropower generation, agricultural and urban use, and to satisfy environmental constraints. One of the most successful techniques has been Dynamic Programming [Bellman and Dreyfus, 1959] and various authors have applied the methodology to water management. Unfortunately, from the very beginning it was apparent that an increase of the dimensionality of the problem, i.e. an addition of reservoirs, caused an exponential increase in the time required to find a solution. This problem was named the "curse of dimensionality" by Bellman and prevented the application of the methodology to real world water systems consisting of more than two or three reservoirs. Recently, Bertsekas and Tsitsiklis [1996] have proposed a methodology, named *neuro-dynamic programming*, based on the functional approximation of the Bellman function using Artificial Neural Networks (ANNs). The ANN-based approximation can be obtained exploring the discretisation grid of the search space with a lower resolution, thus reducing the time required to solve one step of the Bellman equation.

In this paper we present an application of this methodology to the solution of the problem of optimal water management. We have implemented an extension to the Successive Approximation Algorithm which has already been adapted to the case of reservoir management policy design [Piccardi and Soncini-Sessa, 1991]. We finally report some preliminary experimental tests

2. THE PROBLEM

Since the first pioneering works by Maas [1962], the problem of the management of a regulated lake has been represented with a feedback control scheme with feedforward compensation (Figure 1).

The control policy, which is the key element of the scheme, returns the volume u_t to be released from the reservoir, once the current storage value s_t is known. When feedforward compensation is present the policy also depends upon the vector I_t, which represents the meteorological information

and the catchment state. Both these systems are affected by a stochastic disturbance ε_t.

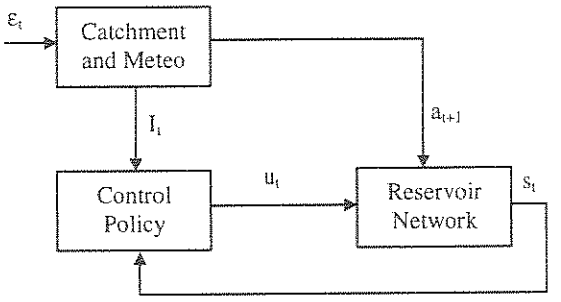

Figure 1. Closed loop control scheme with feed-forward compensation.

According to the chosen modelling representation, the components of the vector I_t can be, e.g., the piezometric head of groundwater, the reservoir inflow during the 24 hours preceding the release decision (a_{t+1}); the stochastic disturbance ε_t can be, e.g., the atmospheric pressure, the solar radiation, the rainfall.

The control policy can be determined as the solution of an optimal control problem defined as follows. The reservoir is represented by the mass conservation equation:

$$s_{t+1} = s_t + a_{t+1} - r_{t+1}(s_t, u_t, a_{t+1}) \tag{1}$$

where r_{t+1} is the actual release in the interval $[t, t+1)$.

The meteorological system and the catchment are the most difficult components to model, because of the complexity of the meteorological and hydrological processes. Very often only the catchment is considered and the reservoir inflow a_{t+1} is represented by simple stochastic autoregressive models of order q.

$$a_t = \chi_t(a_{t-1}, a_{t-2}, \ldots, a_{t-q}, \varepsilon_t) \tag{2}$$

where ε_t is a white gaussian noise. Here

$$I_t = | a_{t-1}, a_{t-2}, \ldots, a_{t-q} |.$$

The model of the whole dynamic system, composed of the meteorological system, the catchment and the reservoir, can thus be represented in the compact vectorial equation:

$$x_{t+1} = f_t(x_t, u_t, \varepsilon_{t+1}) \tag{3}$$

where x_t is the state vector, composed by the state variables in s_t and I_t. Because of climate periodicity, the function $f_t(\cdot,\cdot,\cdot)$ is periodic of period T equal to one year.

During the system evolution, the state transition from x_t to x_{t+1} can produce an instantaneous cost, expressing the lack of fulfillment of management objectives, computed as the weighted sum of the costs associated with the k objectives:

$$g_t = \sum_{j=1}^{k} w^j g_t^j \tag{4}$$

where w^j is the weight of the j-th objective. Also the step costs are periodic with period T.

We define as *policy* the infinite sequence $p = \{m_0, m_1, \ldots\}$ of periodic functions of period T where:

$$u_t = m_t(x_t) \tag{5}$$

The optimal control problem is to find the policy that minimises a function of the costs in the future, over an infinite horizon. Using the Laplace criterion (use of the expected value operator on the disturbance ε), given an initial state x_0 and a discount rate α for the future costs, the cost function of a given policy p is defined as:

$$J(x_0, p) = \lim_{h\to\infty} \sum_{t=0}^{h} E_{\varepsilon_1 \ldots \varepsilon_{h+1}} [\alpha^t g_t(x_t, u_t, \varepsilon_{t+1})]$$

The optimal control problem is therefore solved when we find the policy p^o which minimises:

$$J(x_0) = \min_{p} J(x_0, p) \tag{6a}$$

subject to:

$$x_{t+1} = f_t(x_t, u_t, \varepsilon_{t+1}) \tag{6b}$$

$$\varepsilon_t \sim \Phi_t(\varepsilon_{t+1} \mid x_t, u_t) \tag{6c}$$

$$x_t \in S_t \quad u_t \in U_t(x_t) \quad \varepsilon_t \in D_t \tag{6d}$$

$$u_t = m_t(x_t) \tag{6e}$$

where S_t, U_t, D_t are the discretised domains of the state, control and disturbance.

3. SOLUTION BASED ON STOCHASTIC DYNAMIC PROGRAMMING

The solution of the optimal control problem (6a-6e) by SDP is based on the evaluation of the optimal cost-to-go, which is defined as the cost that one would have to pay if the system would be initially in state x_{t+1} and the system's future trajectory would be obtained applying optimal control decisions in every state transition. We name this cost $H_{t+1}^o(x_{t+1})$. If the optimal cost-to-go would be known for every value of x_{t+1}, the optimal decision $m_t^o(x_t)$ at time t would be easily found minimising

the expected value of the present cost and the discounted optimal cost-to-go:

$$m_t^o(x_t) = \arg\min_{u_t} \mathrm{E}_{\varepsilon_{t+1}}[g_t(x_t, u_t, \varepsilon_{t+1}) + \alpha H_{t+1}^o(x_{t+1})] \quad (7)$$

The optimal cost-to-go associated with the present state is therefore given by the following recursive equation:

$$H_t^o(x_t) = \min_{u_t} \mathrm{E}_{\varepsilon_{t+1}}[g_t(x_t, u_t, \varepsilon_{t+1}) + \alpha H_{t+1}^o(x_{t+1})] \quad (8)$$

which is known as the *Bellman equation* and its solution is the *Bellman function.*

Under the previous hypotheses, it can be shown that the Bellman function is a periodic function, of period T, which can be obtained using the Successive Approximations Algorithm (SAA) [Bertsekas, 1995] that, proceeding backwards in time from T to 1, solves the recursive equation (8) verifying the constraints (6b-6e).

To determine the right hand side of equation (8), the algorithm, for each value of x_t, must explore all the possible values of u_t and of ε_t. Since this algorithm operates on a discrete search space, we have always implicitly assumed that the domains of u, x and ε were discrete. Actually, it is up to the system analyst to find a satisfactory discretisation of the continuous domains of these variables. The choice of the discretisation is fundamental since it reflects on the algorithm complexity which is combinatorial in the number of states, controls and in their discretisations. If we assume to have n states, each one discretised into N classes, the computational cost of SDP is proportional to:

$$N^n \times T \quad (9)$$

where T is the number of time steps.

In other words, if we increase the resolution of the discretisation, thus enhancing the adherence of our model to the real world, or if we consider more controls and states, to describe more complex reservoir networks, it may happen that the time required to compute a policy becomes excessively long.

Many methods have been devised to overcome this limitation. Georgakakos and Marks [1987] and Georgakakos [1989] proposed the Extended Linear Quadratic Gaussian method, an approach based on Pontriagyn's Maximum principle which is not affected by the dimensionality problem, but requires the cost function to be a quadratic function. Another approach is the one proposed by Nardini et al. [1994]. They developed a heuristic control scheme, named Partial Open Loop Feedback Control, based on the substitution of the off-line control problem with a succession of simpler problems, which is effective in presence of a detailed description of the stochastic part of the water system.

In the following, we introduce a new approach based on neuro-dynamic programming [Bertsekas and Tsitsiklis, 1996] which has the advantage of retaining the ability of SDP to deal with highly non-linear problems, while reducing the algorithm complexity thanks to the approximation of the Bellman functions via ANNs.

4. SOLUTION BASED ON NEURO-DYNAMIC PROGRAMMING

As previously seen, the main problem with SDP lies in the dimensions of the search space. If it is too big, it is nearly impossible to find a solution in a reasonable time. Unfortunately water systems with three or more reservoirs are quite common and each reservoir is modelled with one state variable. It is therefore very easy to come to a point where the discretisation grid of the state variables must be very coarse in order to have a computationally solvable problem that the resulting policy is practically unusable.

Another critical factor is the requirement of memory space: for a network with three reservoirs, each state discretised on a grid with 1000 points, under the simplifying assumption that the reservoir inflows are described by gaussian noise (which does not add to the state dimensions), one would need 1.14 Terabytes of memory space to store the Bellman function values in single precision only.

A solution to overcome these limitations is to use an approximation $\tilde{H}$ of the Bellman Function $H_t^o(\cdot)$ to represent the behaviour of the original function interpolating from a limited subset $\bar{S}_t$ of points extracted from discretisation grid S_t, so that $x_t \in \bar{S}_t \subset S_t$. Since computing a point of $H_t^o(\cdot)$ is computationally very expensive, in terms of both CPU time and memory space, reducing the number of computed points will be extremely beneficial.

In the next sections, first we introduce Artificial Neural Networks as function approximators, then we explain how the Bellman function approximations can be used in an algorithm that differs only slightly from the original SAA.

4.1. Approximating the Bellman Function

Among various function approximation schemes we are particularly interested in multilayer feed-forward networks, as they have been shown to be universal approximators [Hornik 1989, Kreinovich 1991]. This means that an ANN can approximate any function to any desired degree of accuracy

provided that a sufficient number of hidden units are used, and thus we can approximate a highly nonlinear map $H(x)$, such as the Bellman function, where x is a vector, with a feed-forward network $\tilde{H}(x,\vartheta)$, where ϑ is the vector of weights on the arcs connecting the network layers.

The objective is to find a network structure (number of hidden layers, number of neurons) so that the vector ϑ can be efficiently computed, while retaining a "good" approximation ability, thus enabling $\tilde{H}(x,\vartheta)$ to be a *compact* representation of $H(x)$.

Note that the dimension of vector ϑ is equal to $r = s(n+2)+1$ where s is the number of neurons and n the dimension of the state. Thus, to store T approximations of a Bellman function, we only need to store rT values. In the case of the network with three reservoirs, using a feedforward ANN with 3 neurons in the hidden layer (which are enough to approximate the Bellman functions in our experimental cases), leads to the requirement of only 216 Kilobytes of memory space, as opposed to the 1.14 Tb with traditional SDP.

The improvement is not so remarkable when we deal with CPU time, since ANNs must be trained.

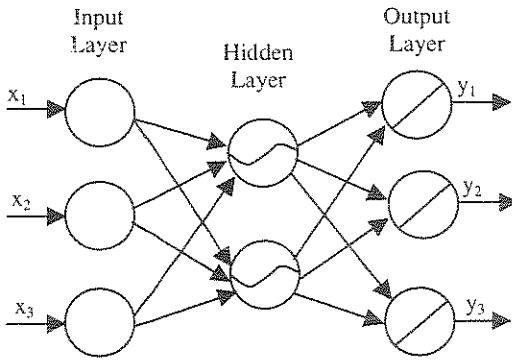

Figure 2. A typical feedforward network.

4.2. Training the Bellman Function Approximations

In a feedforward ANN neurons are organised in layers: the input layer is directly connected with the inputs, the output layer takes the outputs of the hidden layer (one, or more) and produces the network output.

In Figure 2 we have represented a network with three inputs and three outputs, but the Bellman function approximators will always have n inputs, where n is the number of state variables, and a single output (the cost-to-go value).

The input layer (the empty token in Figure 2) simply distributes the input values to the hidden layer weighting the importance of the connections. The weighted inputs are then processed by each node in the hidden layer thanks to an *activation function*. The output of the hidden layer is either passed on to the next hidden layer or sent to the output nodes where it is weighted and processed by a (usually) linear activation function. When all the activation functions are linear, also in the hidden layers, the ANN can be reduced to a linear filter and can be trained using a standard least squares algorithm. ANN show their ability to approximate highly non-linear functions when the activation functions are non-linear. We have chosen to use a hyperbolic tangent as activation function. Unfortunately the least squares algorithm does not work anymore and therefore the backpropagation algorithm has been invented by [Rumelhart et al. 1986]. It minimises the output error of the network (10), given by the sum of squared errors between the target t_o and the network output y_o:

$$E^P = \frac{1}{2}\sum_o (t_o - y_o)^2 \tag{10}$$

The error is minimised computing its derivative with respect to the weights and then applying the forward and backward passes of the backpropagation algorithm to express the weight gradient as a function of the network inputs for each layer.

In the backpropagation algorithm the main problem is the descent of the weight gradient and research has focused on the development of gradient descent algorithms which would converge quickly and avoiding local minima. Most of the time required training a network is spent in these computations, where the trade-off is between accuracy and computational complexity, since most accurate algorithms require the inversion of the Jacobian and the Hessian of the weight matrices of considerable dimensions. Currently we have implemented the Levenberg-Marquardt algorithm [Hagan and Menhaj, 1994] which has been designed to approach the speed of second-order methods without having to compute the Hessian.

4.3. The NDP Algorithm

Once an approximation architecture $\tilde{H}(x,\vartheta)$ of $H(x)$ has been found, the sub-optimal policy $\tilde{m}_t(x_t)$ is given by [Bertsekas and Tsitsiklis, 1996]:

$$\tilde{m}_t(x_t) = \arg\min_{u_t} \mathrm{E}_{\varepsilon_{t+1}}[g_t(x_t,u_t,\varepsilon_{t+1}) + \alpha\tilde{H}_{t+1}(x_{t+1},\vartheta_{t+1})] \quad \forall x_t \in \bar{S}_t \subset S_t \tag{11}$$

Comparing equation (11) with (7), it appears that $\tilde{H}_{t+1}(x_{t+1})$ must be trained using $H^0_{t+1}(x_{t+1})$ as

target and the vector x_{t+1} as the pattern. We remark that the original Bellman function is not available, but we can exploit the recursive nature of the Bellman equation to generate the Bellman functions needed to train their approximations thanks to the *approximate DP formula*:

$$\hat{H}_t(x_t) = \min_{u_t} \mathrm{E}_{\varepsilon_{t+1}}[g_t(x_t, u_t, \varepsilon_{t+1}) + \alpha \tilde{H}_{t+1}(x_{t+1}, \vartheta_{t+1})] \quad (12)$$

The left-hand side of (11) is an approximate cost-to-go function, which can be used to train $\tilde{H}_t(x_t, \vartheta_t)$, which, in turn, will be used in (11) to obtain $\hat{H}_{t-1}(x_{t-1})$. It can be proven formally that if the approximation architecture is "good enough", then $\tilde{H}_t$ is a close approximation of the optimal cost-to-go function H_t^o. Bertsekas and Tsitsiklis [1996] also show that in some particular cases, for some values of the discount rate α, there is still the possibility for algorithm divergence. Such a situation can be avoided by limiting the class of approximators, but also by limiting their power and ease of use.

The algorithm is therefore a simple rewriting of the orginal SAA:

Initialisation. The current algorithm iteration index j is set to 0. Initialise $H_0^{<0>}(\cdot) = 0$ for each state value. Train an ANN $\tilde{H}_T(x_T, \vartheta_T)$ using the discretisation grid of x_{t+1} as the pattern and the identically null function $H_0^{<0>}(\cdot)$ as the target.

Main loop. For each algorithm iteration j compute backwards in time, for t from T-1 down to 0, T functions $\hat{H}_t^{<j>}(\cdot)$ using equation (11). At each time step, after having obtained $\hat{H}_t^{<j>}(\cdot)$, compute its approximation $\tilde{H}_t^{<j>}(\cdot, \vartheta_t)$ training the ANN. When $t = 0$, check if an appropriate convergency criterion, measuring the distance between two Bellman functions at successive iterations, has been satisfied. If not, increment the iteration index j and go back to the beginning of Step 2 after having set $H_0^{<j+1>}(\cdot) = H_T^{<j>}(\cdot)$.

5. PRELIMINARY RESULTS

Currently, the algorithm we have presented in this paper has been applied only to some test cases to verify its correct functioning and to get some first results to understand how to extend its application to real world cases.

A first test was designed to verify the convergence of the algorithm and to obtain some data on its theoretical performance. The test case was a reservoir network with a single reservoir, fed by a catchment described as white gaussian noise, and with an agricultural district with a given water demand, constant over the optimisation period. One step of the SDP algorithm, that is, the evaluation of the right hand side of equation (8) for all the possible values of x_t, given the cardinality of U_t, $card(U_t) = 7$, $card(D_t) = 10$, $card(S_t) = 17$, took 0.97 seconds. One step of the NDP algorithm took 12.41 seconds, of which 0.29 seconds were spent to evaluate the right hand side of (8) for a given x_t, while in the SDP case, the required time was only 0.058 seconds. Another 0.45 seconds were needed for each x_t to train the ANN which approximated the Bellman function obtained in the SDP case with an error less than 10^{-2}. Note that NDP was applied using the same discretisation of the state space of the SDP case. The tests were performed on a Pentium II at 350 Mhz, running under Linux.

We then performed some tests to understand how the training time for an ANN varies with the pattern dimension, which depends on the number of state variables (the dimension of the independent variable vector x in a function $y = f(x)$) and the number of points in the discretisation classes of each component of the vector. These results are reported in Table 1.

Table 1. Training times and approximation errors measured in the training of the functions $y = ln(x)\text{-}sin(x)$ (1 state variable), $y = ln(x) - sin(y)$ (2 variables), $y = ln(x) - sin(y+z)$ (3 variables), $y = ln(x+y)\text{-}sin(z+w)$ (4 variables).

Pattern size		*Training time* (seconds)	*Approximation error*
number of state variables	number of discretisation classes		
1	10	1.3	1.00E-05
1	20	2.06	1.00E-02
1	40	3.39	1.00E-02
1	80	6.2	1.00E-03
2	40	5.01	1.00E-02
3	40	8.01	1.00E-03
4	40	9.89	1.00E-02

The results show that NDP becomes especially interesting only when the number of states increases and the discretisation grid is coarser. We have therefore applied NDP to a test case with two reservoirs, where the state discretisation of each storage was 17 classes. While it was still a scaled-down test case, the gap between NDP and SDP was closing down, thanks to the reduction of the state grid to 8 discretisation classes in the NDP

case. We measured an average time per iteration of 8 seconds in the SDP case, against 47.36 seconds for the NDP case. We are currently experimenting NDP on a real world case, the synthesis of management policies for the Piave water system. No results are yet available as we write, since we are still setting up the experimental framework, but we expect to notice a considerable reduction in computing time, given that there are three state variables, with their discretisation classes equal to 102, 41 and 96 points. Sampling those classes, taking one point out of four, will allow to browse a search space of 6000 points in the NDP case against the 401472 points of the SDP case.

6. CONCLUSIONS

An approach to the management of reservoir networks based on neuro-dynamic programming has been presented. Neuro-dynamic programming allows to reduce the amount of memory needed to store the Bellman functions during the solution of an optimal control problem. It also reduces the computation time when the state space, used as training pattern, is sampled with a coarser grid, while the ANN, which approximates the Bellman function, still manages to maintain a good approximation performance.

The first results are promising, but there is space for more research, especially on the efficient sampling of the discretised state space, in order to obtain the most efficient approximation of the Bellman function.

7. REFERENCES

Bellman, R.E., and Dreyfus, S.E., Functional approximations and dynamic programming, *Mathematical Tables and Other Aids to Computation*, 13, pp. 247–251, 1959.

Bertsekas, D.P., *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, MA, 1995.

Bertsekas, D.P., and J.N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.

Georgakakos, A.P., and Marks, D.H., A new method for real-time operation of reservoir systems, *Water Resour. Res.*, 23(7), pp. 1376–1390. 1987.

Georgakakos, A.P., Extended Linear Quadratic Gaussian Control for the real-time operation of reservoir systems, in *Dynamic Programming for Optimal Water Resources Systems Analysis*, A. Esogbue, ed., Prentice Hall Publishing Company, NJ, pp. 329–360, 1989.

Hagan, M.T., and M. Menhaj, Training feedforward networks with the Marquardt algorithm, *IEEE Transactions on Neural Networks*, 5(6), pp. 989–993, 1994.

Hornik, K., Multilayer feedforward networks are universal approximators, *Neural Networks*, 2, pp. 359–366, 1989.

Kreinovich, V., Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem, *Neural Networks*, vol.4, pp. 381–383, 1991.

Maas, A., M.M. Hufschmidt, R. Dorfam, H.A. Thomas, S.A. Marglin and G.M. Fair, *Design of Water Resource Systems*, Harvard Univ. Press, 1962.

Nardini, A, C. Piccardi and R. Soncini-Sessa, A decomposition approach to suboptimal control of discrete-time systems, *Optimal Control Applications and Methods*, 15, pp. 1–12, 1994.

Piccardi, C. and R. Soncini-Sessa, Stochastic dynamic programming for reservoir optimal control: dense discretization and inflow correlation assumption made possible by parallel computing. *Water Resour. Res.*, 27(2), pp. 729–741, 1991.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams, Learning internal representations by error backpropagation, in *Parallel Data Processing*, D.E. Rumelhart and J.L. McClelland, eds., vol 1, Cambridge, MA: The MIT Press, pp. 318–362, 1986