Published in the IEEE Transactions on Neural Networks, September 1997, pp. 997-1007

Adaptive Critic Designs

Danil Prokhorov and Don Wunsch[1]

Applied Computational Intelligence Laboratory
Department of Electrical Engineering, Box 43102
Texas Tech University, Lubbock, TX 79409, U.S.A.
e-mail: prokhor@eesun1.ee.ttu.edu
http://www.acil.ttu.edu

Abstract

We discuss a variety of Adaptive Critic Designs (ACDs) for neurocontrol. These are suitable for learning in noisy, nonlinear, and nonstationary environments. They have common roots as generalizations of dynamic programming for neural reinforcement learning approaches. Our discussion of these origins leads to an explanation of three design families: Heuristic Dynamic Programming (HDP), Dual Heuristic Programming (DHP), and Globalized Dual Heuristic Programming (GDHP). The main emphasis is on DHP and GDHP as advanced ACDs. We suggest two new modifications of the original GDHP design that are currently the only working implementations of GDHP. They promise to be useful for many engineering applications in the areas of optimization and optimal control. Based on one of these modifications, we present a unified approach to all ACDs. This leads to a generalized training procedure for ACDs.

[1] The authors gratefully acknowledge support from the Texas Tech Center for Applied Research, Ford Motor Co. and National Science Foundation Neuroengineering Program (Grant # ECS-9413120). We also thank Drs. Paul Werbos and Lee Feldkamp for stimulating and helpful discussions.

1. Origins of adaptive critic designs: reinforcement learning, dynamic programming, and backpropagation

Reinforcement learning has been acknowledged by physiologists since the time of Pavlov [1], and has also been a major focus for the neural network community [2], [3]. At the time of these neural network developments, the existence of backpropagation [4], [5], [6], was considered a separate approach. Developments in the separate field of dynamic programming [7], [8], led to a synthesis of all these approaches. Early contributors to this synthesis included Werbos [9], [10], [11], Watkins [12], [13], and Barto, Sutton and Anderson [14]. An even earlier development by Widrow [15] explicitly implements a critic neural element in a reinforcement learning problem.

To begin tracing these developments, consider the difference between traditional supervised learning and traditional reinforcement learning [16]. The former is a type of error-based learning that was an outgrowth of simple Perceptron [17] or Adaline [18] networks. The latter is a form of match-based learning that applies Hebbian learning [19], and, in its simplest manifestation, is a form of classical conditioning [1]. Meanwhile, dynamic programming was attempting to solve a problem that neither neural network approach could handle. If we have a series of control actions that must be taken in sequence, and we do not find out the quality of those actions until the end of that sequence, how do we design an optimal controller ? This is a much harder problem than simply designing a controller to reach a set point or maintain a reference trajectory. Although dynamic programming can handle both deterministic and stochastic cases, here we illustrate it in a deterministic context. Dynamic programming prescribes

a search tracking backwards from the final step, rejecting all suboptimal paths from any given point to the finish, but retaining all other possible trajectories in memory until the starting point is reached. This can be considered a "smart" exhaustive search in that all trajectories are considered, but worthless ones are dropped at the earliest possible point. However, many trajectories that are extremely unlikely to be valuable are nonetheless retained until the search is complete. The result of this is that the procedure is too computationally expensive for most real problems. Moreover, the backward direction of the search obviously precludes the use of dynamic programming in real-time control.

The other references cited above are to works that recognized the fundamental idea of linking backpropagation with reinforcement learning via a critic network. In supervised learning, a training algorithm utilizes a desired output and, having compared it to the actual output, generates an error term to allow the network to learn. It is convenient to use backpropagation to get necessary derivatives of the error term with respect to training parameters and/or inputs of the network. Here we emphasize this interpretation of backpropagation merely as a tool of getting required derivatives, rather than a complete training algorithm.

Critic methods remove the learning process one step from the control network (traditionally called “action network” or “actor” in ACD literature), so that desired trajectory and action signal are not necessary. The critic network learns to approximate the cost-to-go or strategic utility function (the function J of Bellman equation in dynamic programming) and uses the output of an action network as one of its inputs, directly or indirectly. When the critic network learns, backpropagation of error signals can continue along its input pathway back to the action network. To the backpropagation algorithm, this input pathway looks like just another

synaptic connection that needs weight adjustment. Thus, no desired action signal is needed. What is needed is a desired cost function *J*. However, because of various techniques for stretching out a learning problem over time (e.g., [20], [21]), it is possible to use these methods without even knowing desired *J*, but knowing the final cost and the one-step cost (or its estimate) further referred to as the utility *U*. Thus, some of the architectures we will consider involve time-delay elements.

The work of Barto, et. al. [14], and that of Watkins [12], both feature table-look up critic elements operating in discrete domains. These do not have any backpropagation path to the action network, but do use the action signals to estimate a utility or cost function. Barto et. al. use an adaptive critic element for a pole-balancing problem. Watkins [12] created the system known as Q-learning (the name is taken from his notation), explicitly based on dynamic programming. Werbos has championed a family of systems for approximating dynamic programming [10]. His approach generalizes previously suggested designs for continuous domains. For example, Q-learning becomes a special case of an action dependent Heuristic Dynamic Programming (ADHDP; note the action dependent prefix AD used hereafter) in his family of systems. Werbos goes beyond a critic approximating just the function *J*. His systems called Dual Heuristic Programming (DHP)[23], and Globalized Dual Heuristic Programming (GDHP)[22] are developed to approximate derivatives of the function *J* with respect to the states, and both *J* and its derivatives, respectively. It should be pointed out that these systems do not require exclusively neural network implementations: any differentiable structure suffices as a building block of the systems.

This paper focuses on DHP and GDHP and their AD forms as advanced ACDs, although we start by describing simple ACDs: HDP and ADHDP (Section 2). We provide two new modifications of GDHP that are easier to implement than the original GDHP design. We also introduce a new design called ADGDHP, which is currently the topmost in the hierarchy of ACDs (Section 2.4). We show that our designs of GDHP and ADGDHP provide a unified framework to all ACDs, i.e. any ACD can be obtained from them by a simple reconfiguration. We propose a general training procedure for adaptation of the networks of ACD in Section 3. We contrast the advanced ACDs with the simple ACDs in Section 4. In Section 5, we discuss results of experimental work.

2. Design ladder

2.1. HDP and ADHDP

HDP and its AD form have a critic network that estimates the function J (cost-to-go) in the Bellman equation of dynamic programming, expressed as follows

$$J(t)=\sum_{k=0}^{\infty}\gamma^{k}U(t+k), \tag{1}$$

where γ is a discount factor for finite horizon problems ($0<\gamma<1$), and $U(\cdot)$ is the utility function or local cost. The critic is trained forward in time, which is of great importance for real-time operation. The critic network tries to minimize the following error measure over time

$$\|E_1\|=\sum_{t}E_1^2(t), \tag{2}$$

$$E_1(t)=J(Y(t))-\gamma J(Y(t+1))-U(t), \tag{3}$$

where $Y(t)$ stands for either a vector $R(t)$ of observables of the plant (or the states, if available) or a concatenation of $R(t)$ and a control (or action) vector $A(t)$. (The configuration for training the critic according to (3) is shown in Figure 1a.) It should be noted that, although both $J(Y(t))$ and $J(Y(t+1))$ depend on weights W_C of the critic, we do not account for the dependence of $J(Y(t+1))$ on weights W_C while minimizing the error (2). For example, in the case of minimization in the Least Mean Squares (LMS) we could write the following expression for the weights' update

$$\Delta W_C = -\eta\left(J(Y(t)) - \gamma J(Y(t+1)) - U(t)\right)\frac{\partial J(Y(t))}{\partial W_C}, \tag{4}$$

where η is a positive learning rate[2].

We seek to minimize or maximize J in the immediate future thereby optimizing the overall cost expressed as a sum of all $U(t)$ over the horizon of the problem. To do so we need the action network connected as shown in Figure 1b. To get a gradient of the cost function J with respect to the action's weights, we simply backpropagate $\partial J/\partial J$ (i.e., the constant 1) through the network. This gives us $\partial J/\partial A$ and $\partial J/\partial W_A$ for all inputs in the vector A and all the action's weights W_A, respectively.

In HDP, action-critic connections are mediated by a model (or identification) network approximating dynamics of the plant. The model is needed when the problem's temporal nature does not allow us to wait for subsequent time steps to infer incremental costs. When we are able to wait for this information or when sudden changes in plant dynamics prevent us from using the same model, the action network is directly connected to the critic network. This is called action dependent HDP (ADHDP).

2.2. DHP and ADDHP

DHP and its AD form have a critic network that estimates the derivatives of J with respect to the vector Y. The critic network learns minimization of the following error measure over time

$$\|E_2\| = \sum_t E_2^T(t) E_2(t), \tag{5}$$

where

$$E_2(t) = \frac{\partial J(Y(t))}{\partial Y(t)} - \gamma \frac{\partial J(Y(t+1))}{\partial Y(t)} - \frac{\partial U(t)}{\partial Y(t)}, \tag{6}$$

where $\partial(\cdot)/\partial Y(t)$ is a vector containing partial derivatives of the scalar $(\cdot)$ with respect to the components of the vector Y. The critic network's training is more complicated than in HDP since we need to take into account all relevant pathways of backpropagation, as shown in Figure 2, where the paths of derivatives and adaptation of the critic are depicted by dashed lines.

In DHP, application of the chain rule for derivatives yields

$$\frac{\partial J(t+1)}{\partial R_j(t)} = \sum_{i=1}^{n} \lambda_i(t+1) \frac{\partial R_i(t+1)}{\partial R_j(t)} + \sum_{k=1}^{m} \sum_{i=1}^{n} \lambda_i(t+1) \frac{\partial R_i(t+1)}{\partial A_k(t)} \frac{\partial A_k(t)}{\partial R_j(t)}, \tag{7}$$

where $\lambda_i(t+1)=\partial J(t+1)/\partial R_i(t+1)$, and n, m are the numbers of outputs of the model and the action networks, respectively. By exploiting (7), each of n components of the vector $E_2(t)$ from (6) is determined by

$$E_{2j}(t) = \frac{\partial J(t)}{\partial R_j(t)} - \gamma \frac{\partial J(t+1)}{\partial R_j(t)} - \frac{\partial U(t)}{\partial R_j(t)} - \sum_{k=1}^{m} \frac{\partial U(t)}{\partial A_k(t)} \frac{\partial A_k(t)}{\partial R_j(t)}. \tag{8}$$

---

[2] There exists a formal argument on whether to disregard the dependence of J(Y(t+1)) on W_C [24] or, on the contrary, to account for such a dependence [25]. The former is our preferred way of adapting W_C throughout the paper since the latter seems to be more applicable for finite-state Markov chains [8].

Action dependent DHP (ADDHP) assumes direct connection between the action and the critic networks. However, unlike ADHDP, we still need to have a model network because it is used for maintaining the pathways of backpropagation. ADDHP can be readily obtained from our design of ADGDHP to be discussed in the Section 2.4.

The action network is adapted in Figure 2 by propagating $\lambda(t+1)$ back through the model down to the action. The goal of such adaptation can be expressed as follows:

$$\frac{\partial U(t)}{\partial A(t)} + \gamma \frac{\partial J(t+1)}{\partial A(t)} = 0, \qquad \forall\ t. \tag{9}$$

For instance, we could write the following expression for the weights' update when applying the LMS training algorithm

$$\Delta W_A = -\alpha \left(\frac{\partial A(t)}{\partial W_A} \right)^T \left(\frac{\partial U(t)}{\partial A(t)} + \gamma \frac{\partial J(t+1)}{\partial A(t)} \right), \tag{10}$$

where α is a positive learning rate.

2.3. GDHP

GDHP minimizes the error with respect to both J and its derivatives. While it is more complex to do this simultaneously, the resulting behavior is expected to be superior. We describe three ways to do GDHP (Figures 3, 4, and 5). The first of these was proposed by Werbos in [22]. The other two are our own new suggestions.

Training the critic network in GDHP utilizes an error measure which is a combination of the error measures of HDP and DHP (2) and (5). This results in the following LMS update rule for the critic's weights:

$$\Delta W_C = -\eta_1 \big(J(t) - \gamma J(t+1) - U(t) \big) \frac{\partial J(t)}{\partial W_C} - \eta_2 \sum_{j=1}^{n} E_{2j} \frac{\partial^2 J(t)}{\partial R_j(t) \partial W_C}, \tag{11}$$

where E_{2j} is given in (8), and η_1 and η_2 are positive learning rates.

A major source of additional complexity in GDHP is the necessity of computing second order derivatives $\partial^2 J(t)/\partial R(t)\partial W_C$. To get the adaptation signal-2 (the second term in (11)) in the originally proposed GDHP (Figure 3), we first need to create a network dual to our critic network. The dual network inputs the output J and states of all hidden neurons of the critic. Its output, $\partial J(t)/\partial R(t)$, is exactly what one would get performing backpropagation from the critic's output to its input $R(t)$. Here we need these computations performed separately, and explicitly shown as a dual network. Then we can get the second derivatives sought, by a straightforward but careful backpropagation all the way down through the dual network into the critic network. This is symbolized by the dashed line starting from the encircled 1 in Figure 3.

We have recently proposed and successfully tested a GDHP design with critic's training based on deriving explicit formulas for finding $\partial^2 J(t)/\partial R(t)\partial W_C$ (Figure 4) [28], and, to the best of our knowledge, it is the first published successful implementation of GDHP [34]. While this design is more specialized than the original one, its code is less complex which is an important issue since correct implementation of the design of Figure 3 is not a trivial task. We illustrate how to obtain $\partial^2 J(t)/\partial R(t)\partial W_C$ for the critic's training of this GDHP design in an example below.

Finally, we have also suggested and are currently working on the simplest GDHP design with a critic network as shown in Figure 5 [42]. Here the burden of computing the second derivatives $\partial^2 J(t)/\partial R(t)\partial W_C$ is reduced to the minimum by exploiting a critic network with both

scalar output of the J estimate and vector output of $\partial J/\partial R$. Thus, the second derivatives are conveniently obtained through backpropagation.

We do not perform training of the action network through internal pathways of the critic network of Figure 5 leading from its J output to the input R because it would be equivalent to going back to HDP. We already have high quality estimates of $\partial J/\partial R$ as the critic's outputs in the DHP portion of this GDHP design and therefore use them instead[3]. Thus, the action's training is carried out only by the critic's $\partial J/\partial R$ outputs, precisely as in DHP. However, the J output implicitly affects the action's training through the weights' sharing in the critic. Of course, we do use the critic's internal pathways from its J output to the input R to train the action network in the designs of Figures 3 and 4.

Example

This example illustrates how to calculate the mixed second order derivatives $\partial^2 J(t)/\partial R(t)\partial W_C$ for the GDHP design of Figure 4. We consider a simple critic network shown in Figure 6. It consists of two sigmoidal neurons in its only hidden layer and a linear output J. This network is equivalent to the following function

$$J = w_{35} f_3 + w_{45} f_4 + w_{05} R_0$$

$$= \frac{w_{35}}{\exp(-w_{03}R_0 - w_{13}R_1 - w_{23}R_2) + 1} + \frac{w_{45}}{\exp(-w_{04}R_0 - w_{14}R_1 - w_{24}R_2) + 1} + w_{05}R_0 . \qquad (12)$$

---

[3] This situation is typical when ACDs are used for optimal control. In other application domains where the estimates of $\partial J/\partial R$ obtained from the HDP portion of the design may be of a better quality than those of the DHP portion, the use of these more accurate estimates is preferable [40].

Derivatives $\partial J/\partial R_j$, $j = 0, ..., 2$, are obtained as follows

$$\frac{\partial J}{\partial R_j} = \sum_{i=3}^{4} w_{i5} f_i (1 - f_i) w_{ji} + \delta_{j0} w_{05}, \tag{13}$$

where δ_{j0} is the Kronecker delta. We can get the mixed second order derivatives with respect to the weights of the output neuron as follows

$$\begin{aligned} \frac{\partial^2 J}{\partial R_j \partial w_{i5}} &= f_i (1 - f_i) w_{ji} \\ \frac{\partial^2 J}{\partial R_j \partial w_{05}} &= \delta_{j0} \end{aligned}, \tag{14}$$

where $i = 3, 4$, and $j = 0, ..., 2$. For the hidden layer neurons, the required derivatives are

$$\begin{aligned} \frac{\partial^2 J}{\partial R_j \partial w_{ji}} &= w_{i5} f_i (1 - f_i)[1 + (1 - 2 f_i) w_{ji} R_j] \\ \frac{\partial^2 J}{\partial R_j \partial w_{ki}} &= w_{i5} w_{ji} R_k f_i (1 - f_i)(1 - 2 f_i) \end{aligned}, \tag{15}$$

where $i = 3, 4$, $k = 0, ..., 2$, $j = 0, ..., 2$, and $k \neq j$. Thus, based on (11), we can adapt weights in the network using the following expression

$$\Delta w_{ji} = -\eta_1 \left(J(t) - \gamma J(t+1) - U(t) \right) \frac{\partial J(t)}{\partial w_{ji}} - \eta_2 \sum_{k=0}^{2} \left(\frac{\partial J(t)}{\partial R_k} - \gamma \frac{\partial J(t+1)}{\partial R_k} - \frac{\partial U(t)}{\partial R_k} \right) \frac{\partial^2 J(t)}{\partial R_k \partial w_{ji}}, \tag{16}$$

where the indexes i and j are chosen appropriately. We also assume that either $\partial J(t)/\partial R_0 - \gamma \partial J(t+1)/\partial R_0 - \partial U(t)/\partial R_0 = 0$, or $\partial U(t)/\partial R_0 = 0$ since R_0 is a constant bias term.

The example above can be easily generalized to larger networks.

It is clear that HDP and DHP can be readily obtained from a GDHP design with the critic of Figure 5. The simplicity and versatility of this GDHP design is very appealing, and it prompted

us to a straightforward generalization of the critic of Figure 5 for AD forms of ACDs. Thus, we propose action dependent GDHP (ADGDHP), to be discussed next.

2.4. ADGDHP

As all AD forms of ACDs, ADGDHP features a direct connection between the action and the critic networks. Figure 7 shows adaptation processes in ADGDHP. Although one could utilize critics similar with those illustrated in Figures 3 and 4, we found ADGDHP easier to demonstrate when a critic akin to one of Figure 5 is used. In addition, we gained versatility in that the design of Figure 7 can be readily transformed into ADHDP or ADDHP.

Consider training of the critic network. We can write

$$\frac{\partial J(t+1)}{\partial A_k(t)} = \sum_{i=1}^{n} \lambda_{R_i}(t+1) \frac{\partial R_i(t+1)}{\partial A_k(t)}, \tag{17}$$

$$\frac{\partial J(t+1)}{\partial R_j(t)} = \sum_{i=1}^{n} \lambda_{R_i}(t+1) \frac{\partial R_i(t+1)}{\partial R_j(t)} + \sum_{k=1}^{m} \lambda^*_{A_k}(t) \frac{\partial A_k(t)}{\partial R_j(t)}, \tag{18}$$

where $\lambda_{R_i}(t+1) = \dfrac{\partial J(t+1)}{\partial R_i(t+1)}$,

$\lambda_{A_k}(t+1) = \dfrac{\partial J(t+1)}{\partial A_k(t+1)}$,

$\lambda^*_{A_k}(t) = \dfrac{\partial J(t+1)}{\partial A_k(t)} + \dfrac{\partial U(t)}{\partial A_k(t)}$, and n, m are the numbers of outputs of the model and the action networks , respectively.

Based on (17) and (18), we obtain two error vectors, $E_2^R(t) \in R^n$ and $E_2^A(t) \in R^m$ from (6) as follows:

$$E_{2j}^{R}(t) = \frac{\partial J(t)}{\partial R_j(t)} - \gamma \frac{\partial J(t+1)}{\partial R_j(t)} - \frac{\partial U(t)}{\partial R_j(t)}, \qquad (19)$$

$$E_{2k}^{A}(t) = \frac{\partial J(t)}{\partial A_k(t)} - \gamma \frac{\partial J(t+1)}{\partial A_k(t)} - \frac{\partial U(t)}{\partial A_k(t)}. \qquad (20)$$

As in GDHP, the critic network is additionally trained by the scalar error $E_1(t)$ according to (3). If one applies the LMS algorithm, it results in an update rule similar to (11).

Figure 7 also shows the direct adaptation path $\lambda_A(t+1)$ between the action and the critic networks. We express the goal of action's training as follows:

$$\lambda_A(t) = 0, \forall t. \qquad (21)$$

Similar with what we stated in the Section 2.3 on GDHP, training of the action network is not carried out through the internal pathways of the critic network leading from its J output to the input *A* since it would be equivalent to returning to ADHDP. To train the action network, we use only the critic's $\partial J/\partial A$ outputs so as to meet (21). The goal (21) is the same for all AD forms of ACDs.

3. General training procedure and related issues

This training procedure is a generalization of that suggested in [26], [30], [33], [38], [43], and it is applicable to any ACD. It consists of two training cycles: critic's and action's. We always start with critic's adaptation alternating it with action's until an acceptable performance is reached. We assume no concurrent adaptation of the model network, which is previously trained offline, and any reasonable initialization for W_A and W_C.

In the critic's training cycle, we carry out incremental optimization of (2) and/or (5) by exploiting a suitable optimization technique (e.g., LMS). We repeat the following operations N times:

	for HDP, DHP, GDHP	for ADHDP, ADDHP, ADGDHP
1.0.	Initialize $t=0$ and $R(0)$	Initialize $t=0$, $R(0)$, and $A(0)$
1.1.	$V(t) = f_C(R(t), W_C)$	$V(t) = f_C(R(t), A(t), W_C)$
1.2.	$A(t) = f_A(R(t), W_A)$	$R(t+1) = f_M(R(t), A(t), W_M)$
1.3.	$R(t+1) = f_M(R(t), A(t), W_M)$	$A(t+1) = f_A(R(t+1), W_A)$
1.4.	$V(t+1) = f_C(R(t+1), W_C)$	$V(t+1) = f_C(R(t+1), A(t+1), W_C)$

1.5. Compute $E_1(t)$, $E_2(t)$ from (2) and/or (5), and $\partial V(t)/\partial W_C$, to be used in an optimization algorithm, then invoke the algorithm to perform one update of the critic's weights W_C. For the update example, see equations (4) and (11).

1.6. $t = t + 1$; continue from 1.1.

Here $V(t)$ stands for $J(t)$ or $\lambda_Y(t)$, $f_A(\cdot,\ W_A)$, $f_C(\cdot,\ W_C)$, and $f_M(\cdot,\ W_M)$ are the action, the critic and the model networks, with their weights W_i, respectively.

In the action's training cycle, we also carry out incremental learning through an appropriate optimization routine, as in the critic's training cycle above. The list of operations for the action's training cycle is almost the same as that for the critic's cycle above (lines 1.0 - 1.6). However, we need to use (9) or (21), rather than (2) and/or (5); and $\partial A(t)/\partial W_A$ instead of $\partial V(t)/\partial W_C$ before invoking the optimization algorithm for updating the action's weights W_A (see equation (10) for the update example).

The action's training cycle should be repeated M times while keeping the critic's weights W_C fixed. We point out that N and M are lengths of the corresponding training cycles. They are problem-dependent parameters of loosely specified values. If $M=N=1$ we can easily combine both the cycles to avoid duplicating the computations in lines 1.1-1.4. After the action's training cycle is completed, one may check action's performance, then stop or continue the training procedure entering the critic's training cycle again, if the performance is not acceptable yet[4].

It is very important that the whole system consisting of ACD and plant would remain stable while both the networks of ACD undergo adaptation. Regarding this aspect of the training procedure, we recommend to start the first training cycle of the critic with the action network trained beforehand to act as a stabilizing controller of the plant. Such a pretraining could be done on a linearized model of the plant (see, e.g., in [45]).

Bradtke et al. [26] proved that, in the case of the well-known linear quadratic regulation, a linear critic network with quadratic inputs trained by the recursive least squares algorithm in an ADHDP design converges to the optimal cost. If the regulator always outputs actions which are optimal with respect to the target vector for the critic's adaptation, i.e.

$$A^*(t) = \arg\min_A J(R(t), A(t)) , \tag{22}$$

where $J(R(t),A(t))=\gamma J(R(t+1),A(t+1))+U(R(t),A(t))$, then the sequence $A^*(t)$ is stabilizing, and it converges to the optimal control sequence.

[4] Like many other training procedures, ours also implicitly assumes a sufficiently varied set of training examples (e.g., different training trajectories) repeated often enough in order to satisfy persistent excitation - a property well known in a modern identification and adaptive control literature (see, e.g., [37]).

Control sequences obtained through classical dynamic programming are known to guarantee stable control, assuming a perfect match between the actual plant and its model used in dynamic programming. Balakrishnan et al. [43] suggested to stretch this fact over to a DHP-based ACD for linear and nonlinear control of systems with known models. In their design, one performs a training procedure similar to the above. Each training cycle is continued till convergence of the network's weights (i.e., $N \to \infty$, $M \to \infty$ in the procedure above). It is also suggested to use a new randomly chosen $R(0)$ on every return to the beginning of the critic's training cycle (line 1.6 is modified as follows: $t = t + 1$; continue from 1.0). It is argued that whenever the action's weights converge one has a stable control, and such a training procedure eventually finds the optimal control sequence.

While theory behind classical dynamic programming demands choosing the optimal vector $A^*(t)$ of (22) for each training cycle of the action network, we suggest incremental learning of the action network in the training procedure above. A vector $A(t)$ produced at the end of the action's training cycle does not necessarily match the vector $A^*(t)$. However, our experience [28], [30], [44], [46], along with successful results in [33], [38], [43], indicates that choosing $A^*(t)$ precisely is not critical.

No training procedure currently exists that explicitly addresses issues of an inaccurate or uncertain model $f_M(\cdot, W_M)$. It appears that model network errors of as much as 20% are tolerable, and ACDs trained with such inaccurate model networks are nevertheless sufficiently robust [30]. Although it seems consistent with assessments of robustness of conventional neurocontrol (model reference control with neural networks) [31], [32], further research on robustness of control with ACD is needed, and we are currently pursuing this work.

To allow using the training procedure above in presence of the model network's inaccuracies, we suggest to run the model network concurrently with the actual plant or another model, which imitates the plant more accurately than the model network but, unlike this network, is not differentiable. The plant's outputs are then fed into the model network every so often (usually, every time step) to provide necessary alignments and prevent errors of multiple-step-ahead predictions from accumulating. Such a concurrently running arrangement is known under different names including teacher forcing [35] and series-parallel model [36]. After this arrangement is incorporated in an ACD, the critic will usually input the plant's outputs, rather than the predicted ones from the model network. Thus, the model network is mainly utilized to calculate the auxiliary derivatives $\partial R(t+1)/\partial R(t)$ and $\partial R(t+1)/\partial A(t)$.

4. Simple ACDs versus advanced ACDs

The use of derivatives of an optimization criterion, rather than the optimization criterion itself, is known as being the most important information to have in order to find an acceptable solution. In the simple ACDs, HDP and ADHDP, this information is obtained indirectly: by backpropagation through the critic network. It has a potential problem of being too coarse since the critic network in HDP is not trained to approximate derivatives of J directly. An approach to improve accuracy of this approximation has been proposed in [27]. It is suggested to explore a set of trajectories bordering a volume around the nominal trajectory of the plant during the critic's training, rather than the nominal trajectory alone. In spite of this enhancement, we still expect better performance from the advanced ACDs.

Furthermore, Baird [39] showed that the shorter the discretization interval becomes, the slower the training of ADHDP proceeds. In continuous time, it is completely incapable of learning.

DHP and ADDHP have an important advantage over the simple ACDs since their critic networks build a representation for derivatives of J by being explicitly trained on them through $\partial U(t)/\partial R(t)$ and $\partial U(t)/\partial A(t)$. For instance, in the area of model-based control we usually have a sufficiently accurate model network and well-defined $\partial U(t)/\partial R(t)$ and $\partial U(t)/\partial A(t)$. To adapt the action network we ultimately need the derivatives $\partial J/\partial R$ or $\partial J/\partial A$, rather than the J function itself. But an approximation of these derivatives is already a *direct* output of the DHP and ADDHP critics. Although multilayer neural networks are well known to be universal approximators of not only a function itself (direct output of the network) but also its derivatives with respect to the network's inputs (indirect output obtained through backpropagation) [41], we note that the quality of such a direct approximation is always better than that of any indirect approximation for given sizes of the network and the training data. Work on a formal proof of this advantage of DHP and ADDHP is currently in progress, but the reader is referred to Section 5 for our experimental justification.

Critic networks in GDHP and ADGDHP directly approximate not only the function J but also its derivatives. Knowing both J and its derivatives is useful in problems where availability of global information associated with the function J itself is as important as knowledge of the slope of J, i.e. the derivatives of J [40]. Besides, any shift of attention paid to values of J or its derivatives during training can be readily accommodated by selecting unequal learning rates η_1 and η_2 in equation (11) (see Section 2.3). In Section 2.3 we described three GDHP designs.

While the design of Figure 5 seems to be the most straightforward and beneficial from the viewpoint of small computational expenses, the designs of Figures 3 and 4 use the critic network more efficiently.

Advanced ACDs include DHP, ADDHP, GDHP and ADGDHP, the latter two being capable of emulating all the previous ACDs. All these designs assume availability of the model network. Along with direct approximation of the derivatives of J, it contributes to a superior performance of advanced ACDs over simple ones (see the next Section for examples of performance comparison). Although the final selection among advanced ACDs should be certainly based on comparative results, we believe that in many applications the use of DHP or ADDHP is quite enough. We also note that the AD forms of the designs may have an advantage over not action dependent ones in training recurrent action networks.

5. Experimental Studies

This section provides an overview of our experimental work on applying various ACDs to control of dynamic systems. For detailed information on interesting experiments carried out by other researchers in the field, the reader is referred to [33] and [43].

The first problem deals with a simplified model of a commercial aircraft which is to be landed in a specified touchdown region of a runway within given ranges of speed and pitch angle [22]. The aircraft is subject to wind disturbances that have two components: wind shear (deterministic) and turbulent wind gusts (stochastic). To land safely, an external controller should be developed to provide an appropriate sequence of command elevator angles to the aircraft's

pitch autopilot. Along with actual states of the plane, a controller may also use desired values of the altitude h_c and the vertical speed vh_c supplied by an Instrument Landing System (ILS).

To trade off between closely following the desired landing profile from the ILS when far from the ground, and meeting the landing constraints at the touchdown, one could use the following utility function

$$U(t) = \left(1 - \frac{1}{h(t)}\right)\left(a_1(h(t) - h_c(t))^2 + a_2(vh(t) - vh_c(t))^2\right) + \frac{a_3(vh(t)+2)^2 + a_4(x(t)-150)^2}{h(t)}, \quad (23)$$

where a_i, $i = 1, ..., 4$, are experimentally determined constants, and $h(t)$, $vh(t)$, and $x(t)$ are the actual altitude, vertical speed, and horizontal position of the plane. To avoid a singularity at $h(t)=0$, we treat both terms $1/h(t)$ as fixed to unity whenever $h(t) < 1$ ft.

We found the problem with its original system of constraints not challenging enough since even the non-adaptive PID controller provided in [22] could solve it very well . We complicated the problem by shortening the touchdown region of the runway by 30 percent.

We have compared the PID controller, ADHDP, HDP, DHP, and GDHP for the same complicated version of the autolander problem. Implementation details are discussed in [28], [30], and results are summarized in Fig. 8. The most important conclusion is that in going from the simplest ACD, ADHDP, to the more advanced ACDs one can attain a significant improvement in performance.

We have also applied DHP to control of actual hardware, a ball-and-beam system [44][5]. The goal is to balance the ball at an arbitrary specified location on the beam. We use the recurrent multilayer perceptron for both model and action networks. The model network inputs the current

[5] Although we initially attempted an HDP design, we failed to make it work: its critic was not accurate enough to allow the action's training.

position of the ball, $x(t)$, and the servo motor control signal, the latter being the only output of the action network with a sigmoidal output node. It predicts the next ball position, $x(t+1)$. The action network inputs $x(t)$ from the model network and $x_d(t+1)$, the desired ball position at the next time step. The critic network uses $x(t+1)$ and $x_d(t+1)$ to produce an output, $\partial J(t+1)/\partial x(t+1)$.

We trained the action network off-line using a sufficiently accurate model network trained in the parallel identification scheme [36]. We trained the DHP design according to the training procedure described in Section 3. As the utility $U(t)$, we have used the squared difference between $x(t)$ and $x_d(t)$. Training was performed using the node-decoupled extended Kalman filter (NDEKF) algorithm [31]. The typical training trajectory consisted of 300 consecutive points, with two or three distinct desired locations of the ball. We were usually able to obtain an acceptable controller after three alternating critic's and action's training cycles. Starting with $\gamma=0$ in (6), we moved on to $\gamma=0.6$ and 0.9 for the second and the third critic's cycles, respectively.

Figure 9 shows a sample of performance of the DHP action network when tested on the actual ball-and-beam system for three set points *not* used in training. For comparison, performance of a conventional neurocontroller is also given. This neurocontroller of the same architecture as the action network was trained with the same model network by truncated backpropagation through time with NDEKF [32].

Another experiment to date deals with a nonlinear multiple-input-multiple-output (MIMO) system proposed by Narendra and Mukhopadhyay [45] controlled by HDP and DHP designs [46]. This plant has three states, two inputs and two outputs, and it is highly unstable for small input changes. The maximum time delay between the first control input and the second output is equal

to three time steps. The goal is to develop a controller to track two independent reference signals as closely as possible.

Although Narendra and Mukhopadhyay have explored several control cases, here we discuss only the case of fully accessible states and known plant equations. Thus, instead of the model network, we utilize plant equations within the framework of both ACDs.

The action network inputs the plant state variables, $x_i(t)$, $i=1,...,3$, and the desired plant outputs $y_1^*(t+1)$ and $y_2^*(t+1)$, to be tracked by the actual plant outputs $y_1(t+1)=x_1(t+1)$ and $y_2(t+1)=x_2(t+1)$, respectively. Since we have different time delays for each control input/plant output pair, we used the following utility

$$U[t]=\frac{1}{2}\left\{\left[y_1(t+1)-y_1^*(t+1)\right]^2+\left[y_2(t+2)-y_2^*(t+2)\right]^2+\left[y_2(t+3)-y_2^*(t+3)\right]^2\right\}. \qquad (24)$$

The critic's input vector consists *of* $y_1(t+1)$, $y_1^*(t+1)$, $y_2(t+2)$, $y_2^*(t+2)$, $y_2(t+3)$, $y_2^*(t+3)$. Both the action and the critic networks are simple feedforward multilayer perceptrons with one hidden layer of only six nodes. This is a much smaller size than that of the controller network used in [45], and we attribute our success in training to the NDEKF algorithm.

The typical training procedure lasted three alternations of critic's and action's training cycles (see Section 3). The action network was initially pretrained to act as a stabilizing controller [45], then the first critic's cycle began with$\gamma=0.5$ in (6) on a 300-point trajectory.

Figure 10 shows our results for both HDP and DHP. We continued training both designs until their performance was no longer improving. The HDP action network performed much worse than its DHP counterpart. Although there is still room for improvement (e.g., using a larger network), we doubt that HDP performance will ever be as good as that of DHP. Recently

KrishnaKumar [47] has reported HDP performance better than ours in Figure 10a,b. However, our DHP results in Figure 10c,d still remain superior. We think that this is a manifestation of an intrinsically less accurate approximation of the derivatives of J in HDP, as stated in Section 4.

6. Conclusion

We have discussed the origins of adaptive critic designs as a conjunction of backpropagation, dynamic programming, and reinforcement learning. We have shown ACDs through the design ladder with steps varying in both complexity and power, from Heuristic Dynamic Programming to Dual Heuristic Programming, and to Globalized Dual Heuristic Programming and its action dependent form at the highest level. We have unified and generalized all ACDs via our interpretation of GDHP and ADGDHP. Experiments with these ACDs have proven consistent with our assessment of their relative capabilities.

References

[1] I. P. Pavlov, *Conditional Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex.* London: Oxford University Press, 1927.

[2] S. Grossberg, "Pavlovian pattern learning by nonlinear neural networks," in *Pros. Nat. Acad. Sci.*, 1971, pp. 828-831.

[3] A. H. Klopf, *The Hedonistic Neuron: A Theory of Memory, Learning and Intelligence*. Washington: DC: Hemisphere Press, 1982.

[4] P. J. Werbos, "Beyond regression: New Tools for Prediction and Analysis in the Behavioral Sciences," Ph.D. thesis, Committee on Applied Mathematics, Harvard Univ., Cambridge, MA, 1974.

[5] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting.* Wiley, 1994.

[6] *Backpropagation: Theory, Architectures, and Applications*, Y. Chauvin and D. Rumelhart, Eds. Hillsdale: HJ: LawrenceErlbaum, 1995.

[7] R. E. Bellman, *Dynamic Programming.* Princeton: NJ: Princeton Univ. Press, 1957.

[8] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs: NJ: Prentice-Hall, 1987.

[9] P. J. Werbos, "The Elements of Intelligence," *Cybernetic*, no.3, 1968.

[10] P. J. Werbos, "Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence," *General Systems Yearbook*, vol. 22, pp. 25-38, 1977.

[11] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System Modeling and Optimization (Proc. of the Tenth IFIP Conf., New York, NY, 1981)*, R. F. Drenick and F. Kosin, Eds. NY: Springer-Verlag, 1982.

[12] C. Watkins, "Learning From Delayed Rewards," Ph.D. thesis, Cambridge Univ., Cambridge, England, 1989.

[13] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279-292, 1992.

[14] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike Elements that Can Solve Difficult Learning Control Problems," *IEEE Trans. on Sys., Man, Cybern.*, vol. 13, pp. 835-846, 1983.

[15] B. Widrow, N. Gupta, and S. Maitra, "Punish / Reward: Learning With a Critic in Adaptive Threshold Systems," *IEEE Trans. on Sys., Man, Cybern.*, vol. 3, no. 5, pp. 455-465, 1973.

[16] R. S. Sutton, *Reinforcement Learning*. Boston: MA: Kluwer Academic, 1996.

[17] F. Rosenblatt, *Principles of Neurodynamics.* Washington: DC: Spartan Books, 1962.

[18] B. Widrow and M. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," *Proc. IEEE*, vol. 78, no. 9, pp. 1415-1442, 1990.

[19] D. O. Hebb, *The Organization of Behavior*. New York: NY: Wiley, 1949.

[20] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, vol. 3, pp. 9-44, 1988.

[21] P. J. Werbos, "Backpropagation Through Time: What It Is and How To Do It," *Proc. IEEE*, vol. 78, no. 10, pp. 1550-1560, 1990.

[22] W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds., *Neural Networks for Control*, Cambridge: MA: MIT Press, 1990.

[23] D. A. White and D. A. Sofge, Eds., *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, New York: NY: Van Nostrand Reinhold, 1992.

[24] P. J. Werbos, "Consistency of HDP Applied to a Simple Reinforcement Learning Problem," *Neural Networks*, vol. 3, pp. 179-189, 1990.

[25] L. Baird, "Residual Algorithms: Reinforcement Learning with Function Approximation," in *Proc. 12th Int. Conf. on Machine Learning*, San Francisco, CA, July 1995, pp. 30-37.

[26] S. J. Bradtke, B. E. Ydstie, and A. G. Barto, "Adaptive linear quadratic control using policy iteration," in *Proc. Am. Contr. Conf.,* Baltimore, MD, June 1994, pp. 3475-3479.

[27] N. Borghese and M. Arbib, "Generation of Temporal sequences Using Local Dynamic Programming," *Neural Networks*, no.1, pp. 39-54, 1995.

[28] D. Prokhorov, "A Globalized Dual Heuristic Programming and Its Application To Neurocontrol," in *Proc. World Congress on Neural Networks,* Washington, DC, July 1995, pp. II-389-392.

[29] D. Prokhorov and D. Wunsch, "Advanced Adaptive Critic Designs," in *Proc. World Congress on Neural Networks,* San Diego, CA, September 1996, pp. 83-87.

[30] D. Prokhorov, R. Santiago, and D. Wunsch, "Adaptive Critic Designs: A Case Study For Neurocontrol," *Neural Networks*, vol. 8, no. 9, pp. 1367-1372, 1995.

[31] G. Puskorius and L. Feldkamp, "Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 279-297, 1994.

[32] G. Puskorius, L. Feldkamp, and L. Davis, "Dynamic Neural Network Methods Applied to On-Vehicle Idle Speed Control," *Proc. IEEE*, vol. 84, no. 10, pp. 1407-1420, 1996.

[33] F. Yuan, L. Feldkamp, G. Puskorius, and L. Davis, "A Simple Solution to the Bioreactor Benchmark Problem by Application of Q-learning," in *Proc. World Congress on Neural Networks,* Washington, DC, July 1995, pp. II-326-331.

[34] P. J. Werbos, "Optimal Neurocontrol: Practical Benefits, New Results and Biological Evidence," in *Proc. World Congress on Neural Networks,* Washington, DC, July 1995, pp. II-318-325.

[35] R. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, vol. 1, pp. 270-280.

[36] K. S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4-27.

[37] K. S. Narendra and A. M. Annaswamy, *Stable Adaptive Systems*. Englewood Cliffs: NJ: Prentice-Hall, 1989.

[38] R. Santiago and P. J. Werbos, "A New Progress Towards Truly Brain-Like Control," in *Proc. World Congress on Neural Networks,* San Diego, CA, June 1994, pp. I-27-33.

[39] L. Baird, "Advantage Updating," Wright Laboratory, Wright Patterson AFB, Technical Report WL-TR-93-1146, November 1993.

[40] S. Thrun, *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Boston: MA: Kluwer Academic, 1996.

[41] A. Gallant and H. White, "On Learning the Derivatives of an Unknown Mapping with Multilayer Feedforward Networks," *Neural Networks*, vol. 5, pp. 129-138, 1992.

[42] D. Wunsch and D. Prokhorov, "Adaptive Critic Designs," in *Computational Intelligence: A Dynamic System Perspective*, R. Marks et. al., Eds. IEEE Press, 1995, pp. 98-107.

[43] S. N. Balakrishnan and V. Biega, "Adaptive Critic Based Neural Networks for Control," in *Proc. Am. Contr. Conf.,* Seattle, WA, June 1995, pp. 335-339.

[44] P. Eaton, D. Prokhorov, and D. Wunsch, "Neurocontrollers for Ball-and-Beam Systems," in *Intelligent Engineering Systems Through Artificial Neural Networks 6 (Proc. Conf. Artificial Neural Networks in Engineering)*, C. Dagli et. al., Eds. NY: ASME Press, 1996, pp. 551-557.

[45] K. S. Narendra and S. Mukhopadhyay, "Adaptive Control of Nonlinear Multivariable Systems Using Neural Networks," *Neural Networks*, vol. 7, no. 5, pp. 737-752, 1994.

[46] N. Visnevski and D. Prokhorov, "Control of a Nonlinear Multivariable System with Adaptive Critic Designs," in *Intelligent Engineering Systems Through Artificial Neural Networks 6 (Proc. Conf. Artificial Neural Networks in Engineering)*, C. Dagli et. al., Eds. NY: ASME Press, 1996, pp. 559-565; note misprints in the rms error values.

[47] K. KrishnaKumar, "Adaptive Critics: Theory and Applications," Tutorial at *Conf. Artificial Neural Networks in Engineering (ANNIE'96)*, St. Louis, MO, November 10-13, 1996.

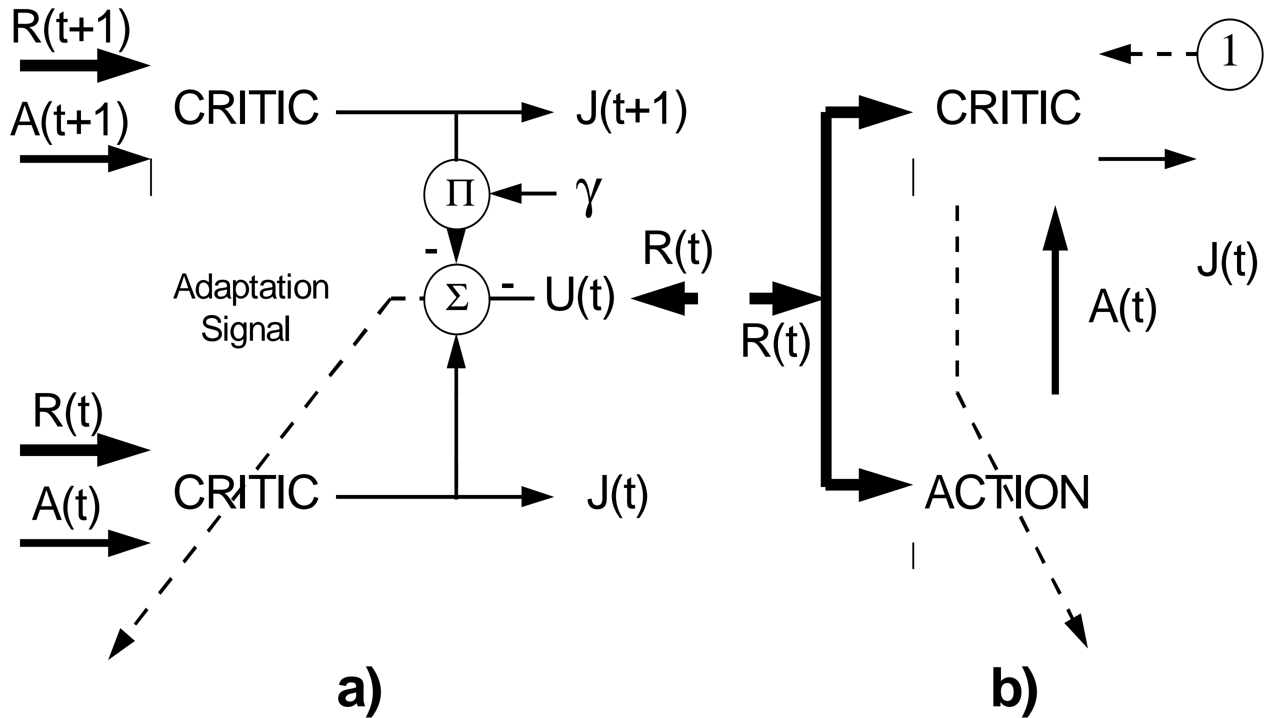


Figure 1

a) Critic adaptation in ADHDP/HDP. This is the same critic network in two consecutive moments in time. The critic's output $J(t+1)$ is necessary in order to give us the training signal $\gamma J(t+1) + U(t)$, which is the target value for $J(t)$.

b) Action adaptation. R is a vector of observables, A is a control vector. We use the constant $\partial J/\partial J = 1$ as the error signal in order to train the action network to minimize J.

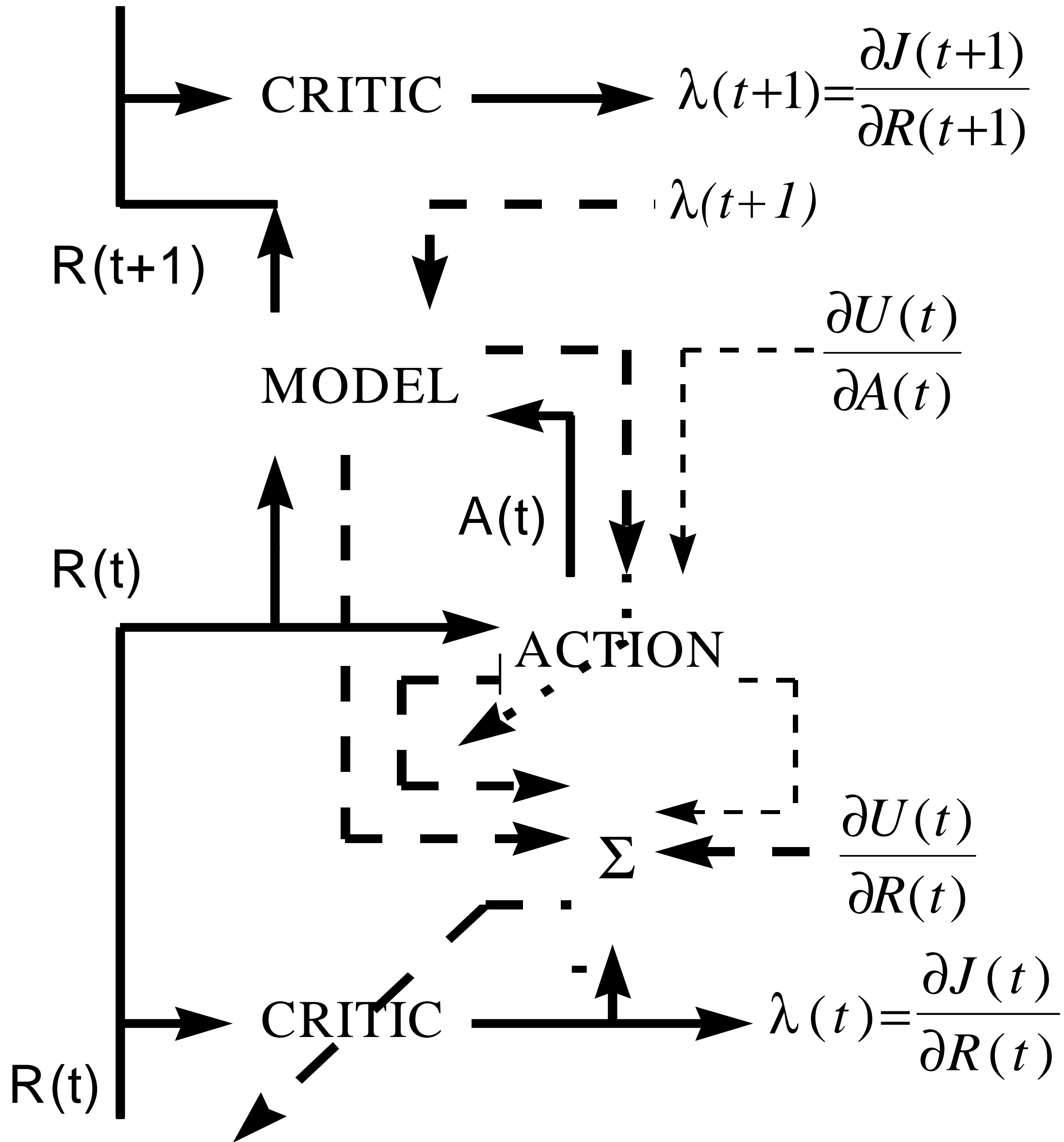


Figure 2

Adaptation in DHP. This is the same critic network shown in two consecutive moments in time. The discount factor γ is assumed to be equal to *1*. Pathways of backpropagation are shown by dashed lines. Components of the vector $\lambda(t+1)$ are propagated back from outputs $R(t+1)$ of the model network to its inputs $R(t)$ and $A(t)$ yielding the first term of (7), and the vector $\partial J(t+1)/\partial A(t)$, respectively. The latter is propagated back from outputs $A(t)$ of the action network to its inputs $R(t)$, completing the second term in (7). This corresponds to the left-hand backpropagation pathway (thicker line) in the Figure. Backpropagation of the vector $\partial U(t)/\partial A(t)$ through the action network results in a vector with components computed as the last term of (8). This corresponds to the right-hand backpropagation pathway from the action network (thinner line) in the Figure. Following (8), the summator produces the error vector $E_2(t)$ used to adapt the critic network. The action network is adapted as follows. The vector $\lambda(t+1)$ is propagated back through the model network to the action network, and the resulting vector is added to $\partial U(t)/\partial A(t)$. Then an incremental adaptation of the action network is invoked with the goal (9).

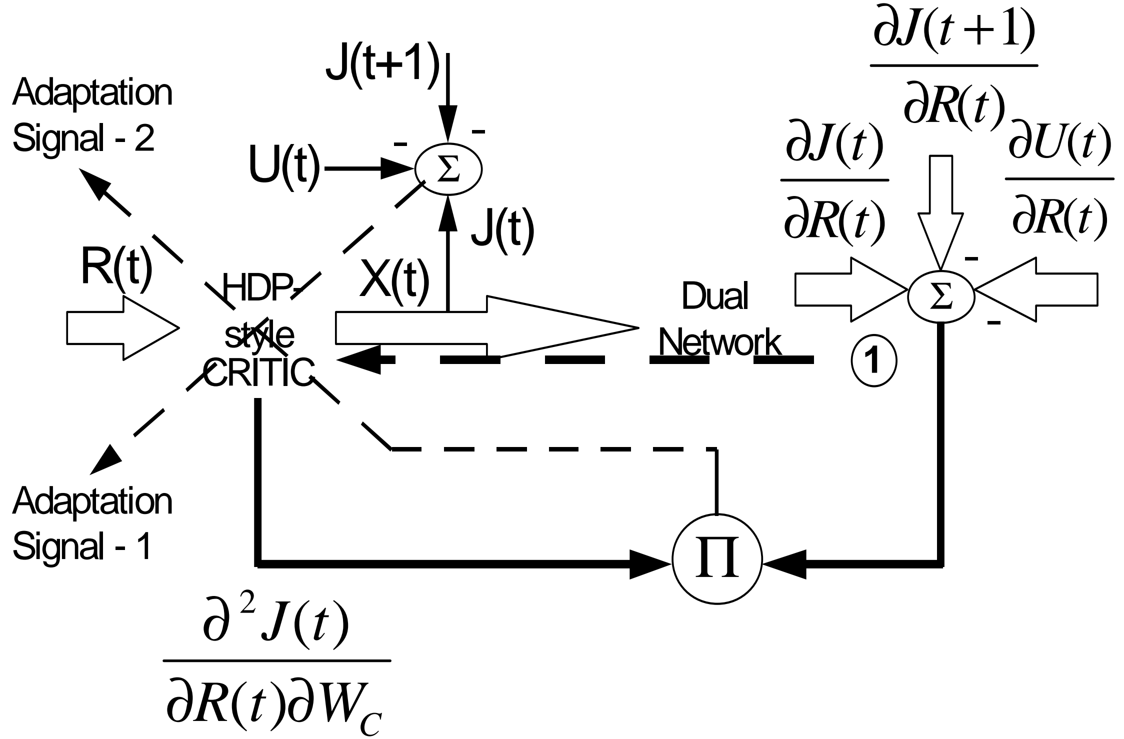


Figure 3

Critic's adaptation in the general GDHP design. X is a state vector of the network. η_1 (Adaptation Signal-1) + η_2 (Adaptation Signal-2) is the total adaptation signal (see the equation (11)). The discount factor γ is assumed to be equal to *1*. According to (3), the summator at the upper center outputs the HDP-style error. Based on (6), the summator to the right produces the DHP-style error vector. The mixed second order derivatives $\partial^2 J(t)/\partial R(t)\partial W_C$ are obtained by finding derivatives of outputs $\partial J(t)/\partial R(t)$ of the critic's dual network with respect to the weights W_C of the critic network itself. (This is symbolized by the dashed arrow that starts from the encircled 1.) The multiplier performs a scalar product of the vector (6) with an appropriate column of the array $\partial^2 J(t)/\partial R(t)\partial W_C$, as illustrated by the equation (16) in the Example.

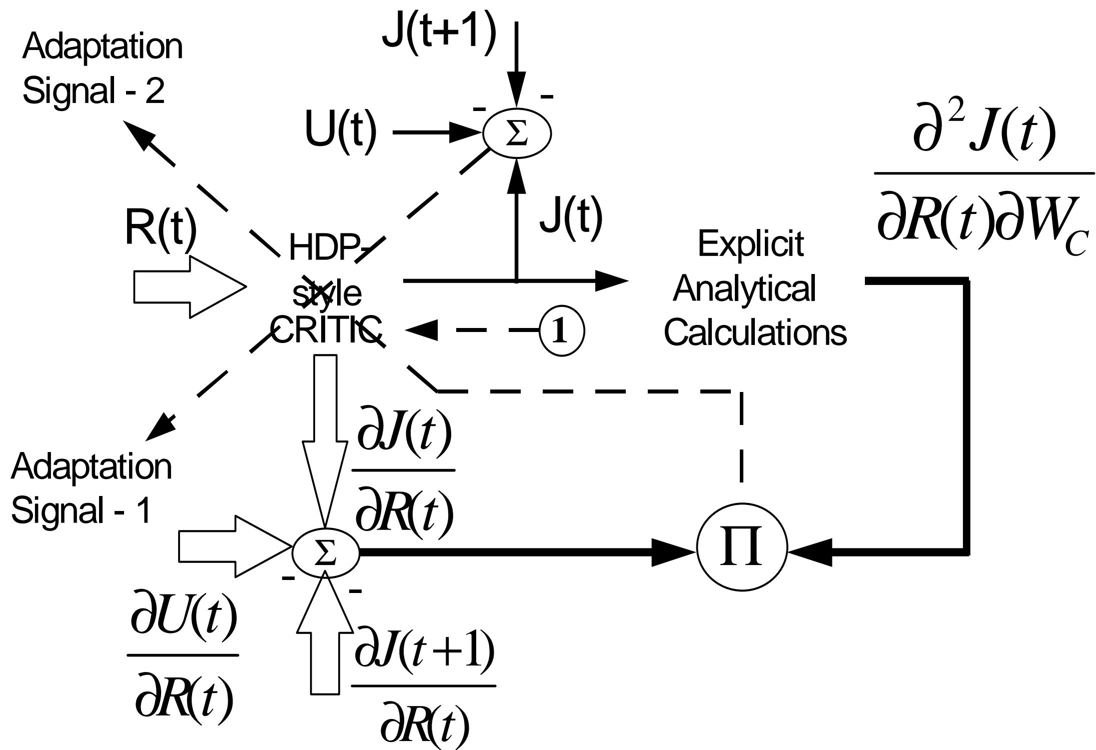


Figure 4
Critic adaptation in our simplified GDHP design. Unlike GDHP in Figure 3, here we use explicit formulas to compute all necessary derivatives $\partial^2 J(t)/\partial R(t)\partial W_C$.

Figure 5. Critic network in a straightforward GDHP design.

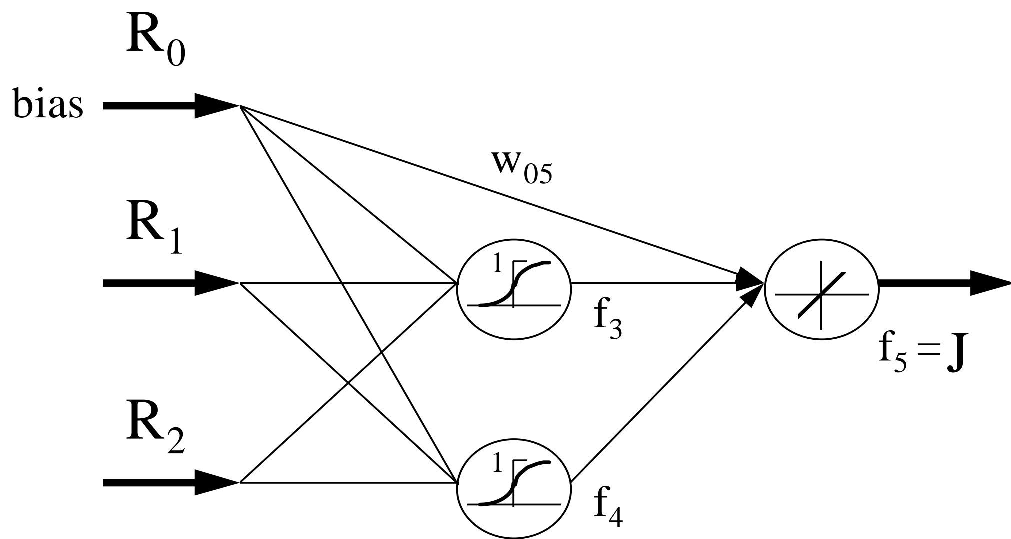


Figure 6
A simple network for the Example of computing the second order derivatives $\partial^2 J(t)/\partial R(t)\partial W_C$ in our GDHP design given in Fig. 4.

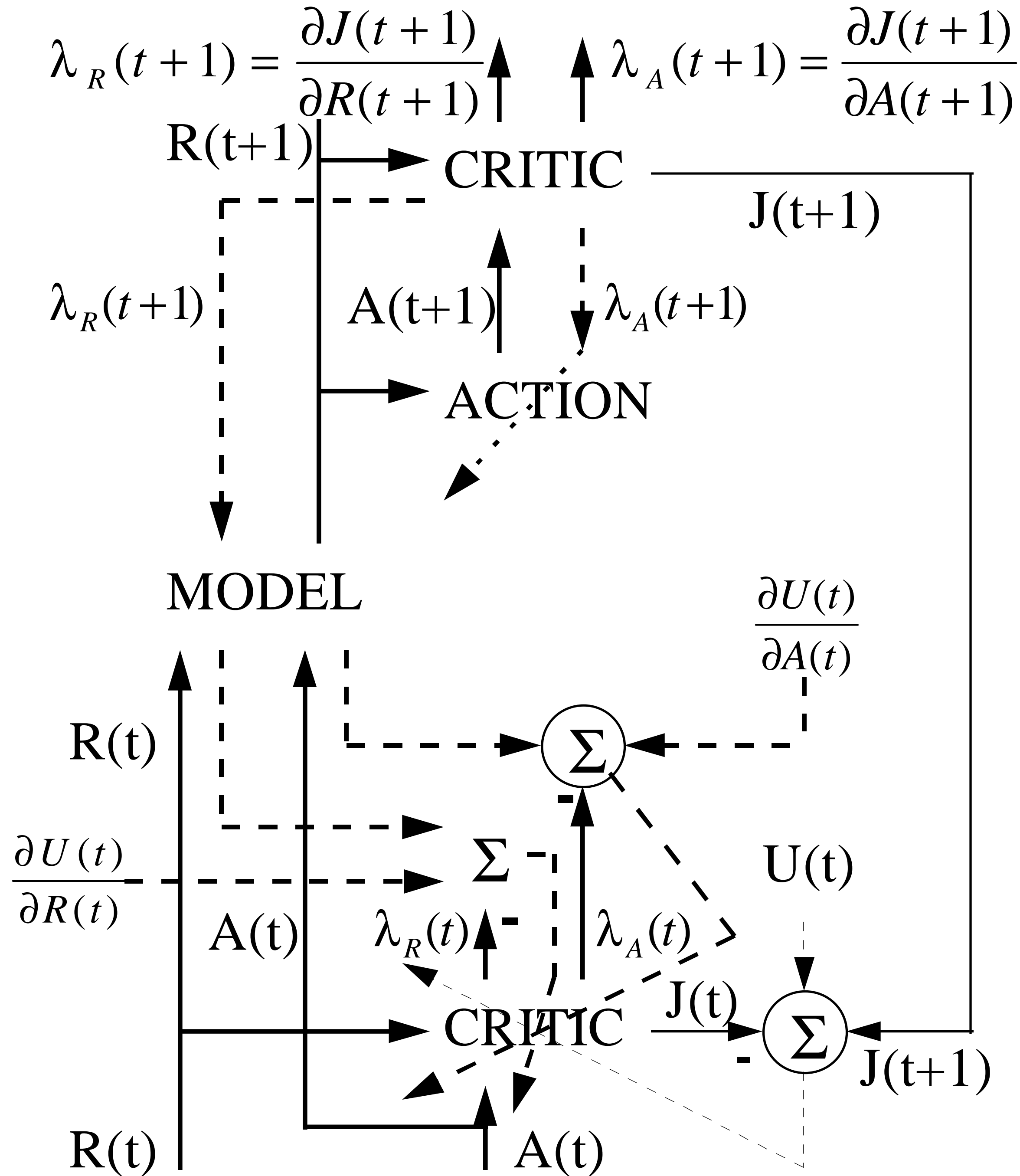
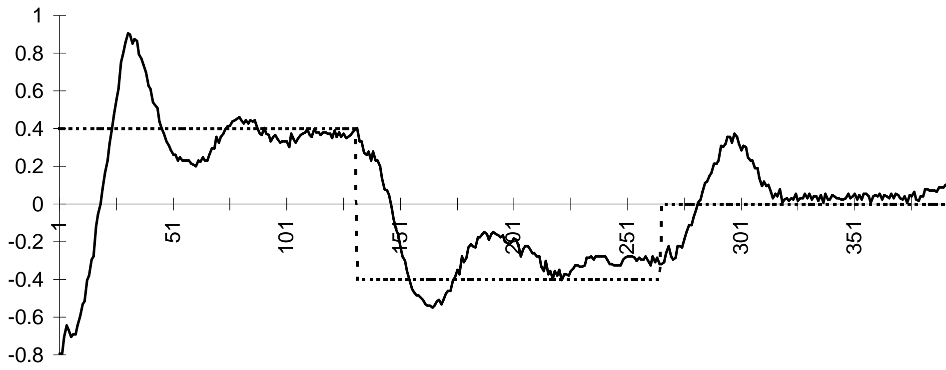


Figure 7

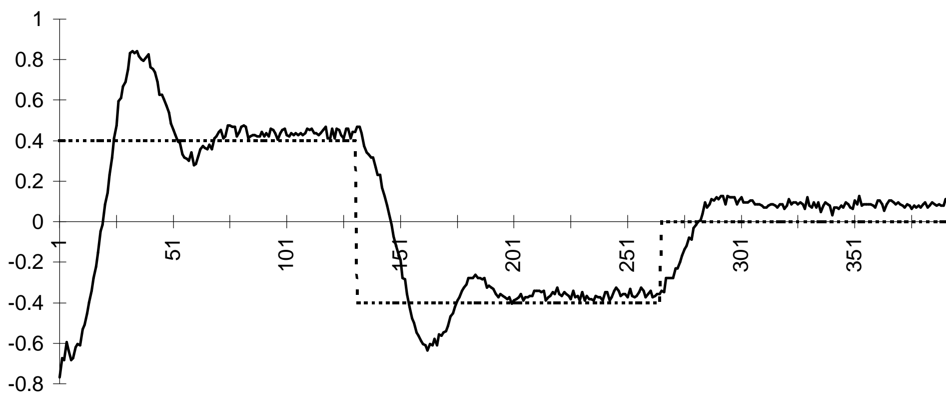
Adaptation in ADGDHP. The critic network outputs the scalar J and two vectors, λ_R and λ_A. The vector λ_R(t+1) propagates back through the model, then it is split in two vectors. One of them goes into the square summator to be added to the vector $\partial U(t)/\partial R(t)$ and to the rightmost term in (18) (not shown). The second vector is added to the vector $\partial U(t)/\partial A(t)$ in another summator. Both of these summators produce two appropriate error vectors $E_2(t)$, as in (19) and (20). According to (3), the right oval summator computes the error $E_1(t)$. Two error vectors $E_2(t)$ and the scalar $E_1(t)$ are used to train the critic network. The action network is adapted by the direct path $\lambda_A(t+1)$ between the critic and the action networks.

Gusts N(0,1.5)		%, out of 600 test trials				
		GDHP	DHP	HDP	ADHDP	PID
Trained with wind shear only	tight success	73	71	50	1	0
	loose success	99	99	98	98	99
Trained with wind shear and wind gusts	tight success	71	70	45	0	0
	loose success	98	98	97	97	98
Average number of training attempts to land		1000	1000	100	100	N/A

Figure 8
Test results of the autolander problem given for one of the most challenging cases where wind gusts were made 50% stronger than in standard conditions. After the ACDs were trained on the number of landings shown, they were tested in 600 more trials, without any adaptation. Although the average training is much longer for GDHP and DHP than for HDP and ADHDP, we could not observe an improvement of performance for either HDP or ADHDP if we continued their training further. Tight success means landing within a shortened touchdown region of the runway (it is the most important characteristic). Loose success means landing within the limits of the standard runway. Similar results were obtained in various other flight conditions.



a)



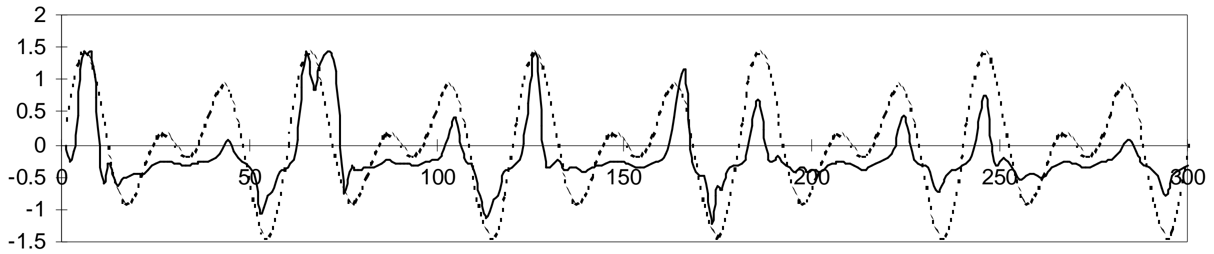
b)

Figure 9

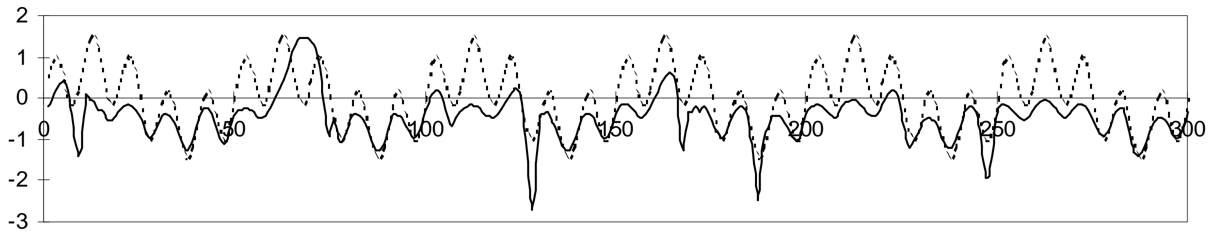
Test results of two neurocontrollers for the ball-and-beam system. Edges of the beam correspond to ±1, and its center is at 0. Dotted lines are the desired ball positions x_d (set points), solid lines are the actual ball trajectory $x(t)$.

a) Conventional neurocontroller trained by the truncated backpropagation through time with NDEKF;

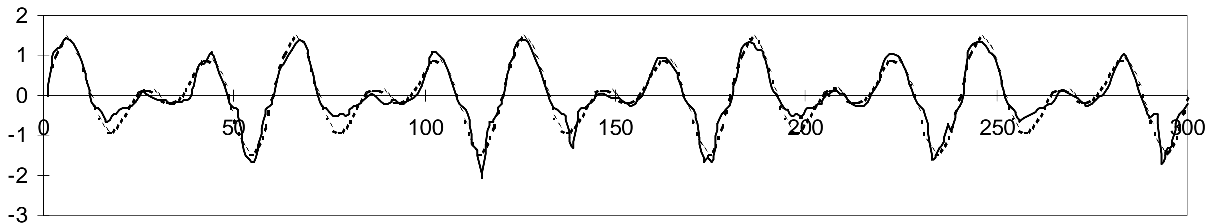
b) DHP action network tested on the same set points as in a).



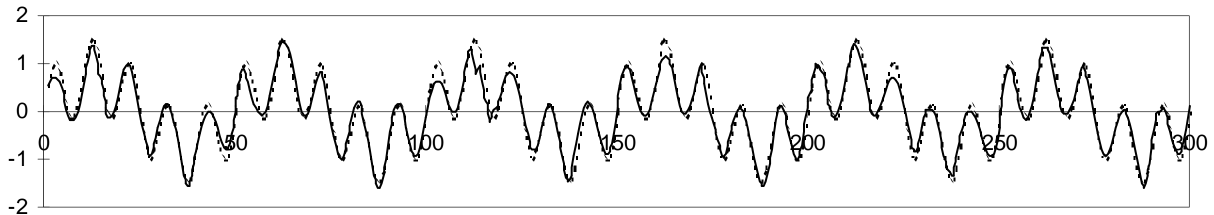
a) $y_2(t)$



b) $y_1(t)$



c) $y_2(t)$



d) $y_1(t)$

Figure 10
Performance of HDP (plots *a* and *b*) and DHP (*c* and *d*) for the MIMO plant. Dotted lines are the reference trajectories y_1^* and y_2^*, solid lines are the actual outputs $y_1(t)$ and $y_2(t)$. The RMS error for DHP is 0.32 vs. 0.68 for HDP.