

To appear as CHAPTER 4 in:

LEARNING & APPROXIMATE DYNAMIC PROGRAMMING
Scaling up to the Real World.

Edited by: Jennie Si, Andy Barto, Warren Powell, and Donald Wunsch

Wiley Interscience, 2004

GUIDANCE IN THE USE OF ADAPTIVE CRITICS FOR CONTROL

GEORGE LENDARIS AND JAMES C. NEIDHOEFER

1. INTRODUCTION

The aim of this chapter is to provide guidance to the prospective user of the Adaptive Critic / Approximate Dynamic Programming methods for designing the action device in certain kinds of control systems. While there are currently various different successful “camps” in the Adaptive Critic community, spanning government, industry, and academia, and while the work of these independent groups may entail important differences, there are basic common threads. The latter include: Reinforcement Learning (RL), Dynamic Programming (DP), and basic Adaptive Critic (AC) concepts.

Describing and understanding the fundamental equations of DP is not difficult. Similarly, it is not difficult to show diagrams of different AC methodologies and understand conceptually how they work. However, understanding the wide variety of issues that crop up in actually applying the AC methodologies is both non-trivial and crucial to the success of the venture. Some of the important tasks include: formulating (appropriately) the problem-to-be-solved; defining a utility function that properly captures/embodies the problem-domain requirements; selecting the discount factor; designing the training “syllabus”; designing training strategies and selecting associated run-time parameters (epoch size, learning rates, etc.); deciding when to start and stop training; and, not the least, addressing stability issues. A brief overview of the three topics listed in the previous paragraph (RL, DP, and AC) is given first, followed by the main body of this chapter, in which selected issues important to successful application of ACs are described, and some approaches to addressing them are presented.

Clearly, while much progress in the development and application of Adaptive Critics has already occurred, much yet remains to be done. The last section of the chapter describes some items the authors deem important to be included in a future research agenda for the Adaptive Critic community.

2. REINFORCEMENT- LEARNING

Reinforcement-Learning (RL) occurs when an agent learns behaviors through trial-and-error interactions with its environment, based on “reinforcement” signals from the environment. In the past ten to fifteen years, the potential of reinforcement-learning has excited the imagination of researchers in the machine learning, intelligent systems, and artificial intelligence communities. Achievement of such potential, however, can be elusive, as formidable obstacles reside in the details of computational implementation.

In a general RL model, an agent interacts with its environment through sensors (perception) and actuators (actions) [13], [6]. Each interaction iteration typically includes the following: the agent receives inputs that indicate the state of the environment; the agent then selects and takes an action, which yields an output; this output changes the state of the environment, transitioning it to a “better” or a “worse” state; the latter are indicated to the agent by either a “reward” or a “penalty” from the environment, and the amount of such reward/penalty has the effect of a “reinforcement” signal to the agent. The behavior of a healthy agent tends to increase the reward part of the signal, over time, through a trial and error learning process. Thorndike’s law of effect has been rephrased in [5], [56] to offer the following definition of reinforcement-learning: “If an action taken by a learning system is followed by a satisfactory state of affairs, then the tendency of the system to produce that particular action is strengthened or reinforced. Otherwise, the tendency of the system to produce that action is weakened.” Reinforcement-Learning differs from supervised-learning mainly in the kind of feedback received from the environment. In supervised-learning, the equivalent of a “teacher” function is available that knows the correct output, a priori, for each of the agent’s outputs, and training/learning is based on output error data. In RL, on the other hand, the agent only receives a more general, composite reward/punish signal, and learns from this using an operating principle of increasing the amount of reward it receives over time. While RL has been implemented in a variety of different ways and has involved other related research areas (e.g., search and planning), in this chapter, we focus on application of the RL ideas to implementing approximate Dynamic Programming, often called Adaptive Critics. We comment that the phrase ‘Adaptive Critic’ was originally coined by Widrow, [61], in a manner that implies learning with a critic. The present authors, on the other hand, prefer that the term ‘adaptive’ in

the phrase refer to the critic’s learning attribute. We note that [59] also uses the term ‘adaptive’ in this latter sense.

3. DYNAMIC PROGRAMMING

Dynamic Programming (DP) [7] provides a principled method for determining optimal control policies for discrete-time dynamic systems whose states evolve according to given transition probabilities that depend on a decision/control u . Simultaneous with a transition from one state (call it $X(t)$) to the next ($X(t + 1)$) under control u , a cost U is incurred [8]. Optimality is defined in terms of minimizing the sum of all the costs to be incurred while progressing from any state to the end state (both, finite and infinite cases are handled). This sum of costs is called ‘cost-to-go,’ and the objective of DP is to calculate numerically the optimal cost-to-go function J^* . An associated optimal control policy is also computed. Fundamental to this approach is Bellman’s Principle of Optimality, which states that: “no matter how an intermediate point is reached in an optimal trajectory, the rest of the trajectory (from the intermediate point to the end) must be optimal.” Unfortunately, the required DP calculations become cost-prohibitive as the number of states and controls become large (Bellman’s “curse of dimensionality”); since most real-world problems fall into this category, approximating methods for DP have been explored since its inception (e.g. see [27]). The DP method entails the use of a Utility function, where the Utility function is crafted (by the user) to embody the design requirements of the given control problem. This function provides the above-mentioned ‘cost’ incurred while transitioning from a given state to the next one. A secondary utility function, known as the Value function (referred to above as the Cost-to-Go function), is defined in terms of the Utility function, and is used to perform the optimization process. [Bellman used the technically correct terminology Value *functional*, but much of our literature uses *function* instead; the latter is used in this chapter.] Once the ‘optimal’ version of the Value function has been determined, then the optimal controller may be designed, e.g., via the Hamilton-Jacobi-Bellman equation.

4. ADAPTIVE CRITICS: “APPROXIMATE DYNAMIC PROGRAMMING”

The Adaptive Critic concept is essentially a juxtaposition of RL and DP ideas. It will be important to keep in mind, however, that whereas

DP calculates the control via the optimal Value Function, the AC concept utilizes an approximation of the optimal Value Function to accomplish its controller design. For this reason, AC methods have been more properly referred to as implementing Approximate Dynamic Programming (ADP). A family (also called “ladder”) of ADP structures was proposed by Werbos in the early 1990’s [59], [60], and has been widely used by others [11], [12], [14], [15], [18], [19], [21], [22], [23], [24], [25], [26], [32], [33], [34], [36], [37], [38], [41], [43], [45], [46]. While the original formulation was based on neural network implementations, it was noted that any learning structure capable of implementing the appropriate mathematics would work. Fuzzy Logic structures would be a case in point; recent examples may be found in [25], [44], [48], [53], [55]. This family of ADP structures includes: Heuristic Dynamic Programming (HDP), Dual Heuristic Programming (DHP), and Global Dual Heuristic Programming (GDHP). There are ‘action dependent’ (AD) versions of each, yielding the acronyms: ADHDP, ADDHP, and ADGDHP. A detailed description of all these ADP structures is given in [43], called Adaptive Critic Designs there; additional details may also be found in [41].

The different ADP structures can be distinguished along three dimensions: 1) The inputs provided to the critic; 2) the outputs of the critic; and 3) the requirements for a plant model in the training process.

4.1. Critic Inputs. The critic typically receives information about the state of the plant (and of a reference model of the plant, where appropriate); in the action dependent structures, the critic is also provided the outputs of the action device (controller).

4.2. Critic Outputs. In the HDP structure, the critic outputs an approximation of the Value Function $J(t)$; in the DHP structure, it approximates the gradient of $J(t)$; and in the GDHP, it approximates both, $J(t)$ and its gradient.

4.3. Model requirements. While there exist formulations that require only one training loop (e.g. [30]), the above ADP methods all entail the use of two training loops: one for the controller and one for the critic. There is an attendant requirement for two trainable function approximators, one for the controller and one for the critic. Depending on the ADP structure, one or both of the training loops will require a *model of the plant*. The controller training loop adapts the function approximator to be an approximately optimal controller (whose outputs are $u(t)$), via maximizing the secondary utility function $J(t)$. Since

a gradient-based learning algorithm is typically used, derivatives' estimates are required for controller training (in the DHP version, these estimates are provided directly from the critic). Adaptation of the function approximation in the critic training loop is based on the consistency of its estimates through time, the exact implicit relationship being a function of the type of critic used and the structure of the primary utility function. In this chapter, we focus on the DHP structure; this structure requires a plant model for both loops. We mention that some view this model dependence to be an unnecessary "expense." The position of the authors, however, is that the expense is in many contexts more than compensated for by the additional information available to the learning/optimization process. We take further motivation for pursuing model-dependent versions from the biological exemplar: some explanations of the human brain developmental/learning process invoke the notion of 'model imperative' [39].

4.4. Model Use in Training Loops. Figure 1 provides a general diagrammatic layout for the ADP discussion. The base components are the action/controller and the plant; the controller receives measurement data about the plant's current state $X(t)$ and outputs the control $u(t)$; the plant receives the control $u(t)$, and moves to its next state $X(t+1)$. The $X(t)$ data is provided to the critic and to the Utility function. In addition, the $X(t+1)$ data is provided for a second pass through the critic. All of this data is needed in the calculations for performing the controller and critic training (the various dotted lines going into the 'calculate' boxes). This training is based on the Bellman Recursion:

$$(1) \quad J(t) = U(t) + \gamma J(t+1)$$

We note that the term $J(t+1)$ is an important component of this equation, and is the reason that $X(t+1)$ is passed through the critic to get its estimate for time $(t+1)$ (see [17] [16] for fuller expansion of the equations involved).

The following is a verbal "walk through" of the six different AC structures, pointing out why and in which loop(s) of each structure a plant model is required. The results are tabulated in Table 1.

HDP: The critic estimates $J(t)$ based directly on the plant state $X(t)$; since this data is available directly from the plant, critic training does not need a plant model for its calculations. Controller training, on the other hand, requires finding the derivatives of $J(t)$ with respect to the control variables, obtained via the chain rule $\frac{\partial J(t)}{\partial u_i(t)} =$

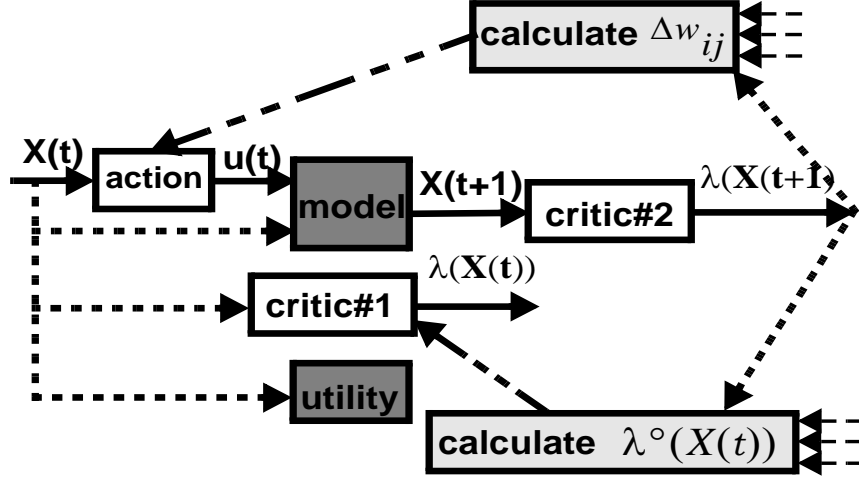


FIGURE 1. General layout of Adaptive Critic structures

$\sum_{j=1}^n \frac{\partial J(t)}{\partial X_j(t)} \frac{\partial X_j(t)}{\partial u_i(t)}$. Estimates of the first term in this equation (derivatives of $J(t)$ with respect to the states) are obtained via Backpropagation through the critic network; estimates for the second term (derivatives of the states with respect to the controls) require a differentiable model of the plant, e.g., an explicit analytic model, a neural network model, etc. Thus HDP uses a plant model for the controller training but not the critic training.

ADHDP (Q-learning is in this category): Critic training is the same as for HDP. Controller training is simplified, in that since the control variables are inputs to the critic, the derivatives of $J(t)$ with respect to the controls, $\left(\frac{\partial X(t)}{\partial u(t)}\right)$, are obtained directly from Backpropagation through the critic. Thus ADHDP uses no plant models in the training process.

DHP: Recall that for this version, the critic directly estimates the derivatives of $J(t)$ with respect to the plant states, i.e. $\lambda_i(t) = \frac{\partial J(t)}{\partial X_i(t)}$. The identity used for critic training is (in tensor notation):

$$\lambda_i(t) = \frac{\partial U(t)}{\partial X_i(t)} + \frac{\partial U(t)}{\partial u_i(t)} \frac{\partial u_j(t)}{\partial X_i(t)} + \lambda_k(t+1) \left[\frac{\partial X_k(t+1)}{\partial X_i(t)} + \frac{\partial X_k(t+1)}{\partial u_m(t)} \frac{\partial u_m(t)}{\partial X_i(t)} \right]$$

To evaluate the right hand side of this equation, a full model of the plant dynamics is needed. This includes all the terms for the Jacobian matrix of the coupled plant-controller system, e.g. $\frac{\partial X_j(t+1)}{\partial X_i(t)}$ and $\frac{\partial X_j(t+1)}{\partial u_i(t)}$. Controller training is much like that in HDP, except that the controller

training loop directly utilizes the critic outputs along with the system model. So, DHP uses models for both critic and controller training.

ADDHP: ADDHP critics use both state and control variables as inputs, and output the gradient of $J(t)$ with respect to both states and controls, $\left(\frac{\partial J(t)}{\partial X(t)}\right)$ and $\left(\frac{\partial X(t)}{\partial u(t)}\right)$. This method utilizes the DHP critic training process, but gets the derivatives needed for controller training directly from the critic's output. Therefore ADDHP uses a plant model for critic training but not for controller training.

GDHP: GDHP critics have state variables as inputs, and they output both $J(t)$ and its gradient with respect to states. Critic training utilizes both the HDP and DHP recursions; controller training as in DHP. Therefore GDHP uses models for both critic and controller training.

ADGDHP: ADGDHP critics have both state and control variables as inputs, and they output both $J(t)$ and its gradient with respect to states and controls. As with GDHP, critic training utilizes both the HDP and DHP recursions, and controller training is as in ADDHP. Therefore ADGDHP uses a model for critic training but not for controller training.

ADP STRUCTURE	Model NEEDED for training of	
	CRITIC	CONTROLLER
HDP		X
ADHDP		
DHP	X	X
ADDHP	X	
GDHP	X	X
ADGDHP	X	

Table 1. Summary of Requirement For Model in Training Loops

5. SOME CURRENT RESEARCH ON ADAPTIVE CRITIC TECHNOLOGY

As part of providing guidance to prospective users of the ADP methods to develop controller designs, we sketch some of the work being done in the area, and provide citations that the reader may find useful.

Andersen and his colleagues at Colorado State University have been working on combining Robust control theory with Reinforcement-Learning methodologies to develop proofs for both, static and dynamic stability, (e.g. [1],[2]). A Reinforcement-Learning procedure has resulted which

is guaranteed to remain stable even during training. In an attempt to speed up the process (which turns out being on the slow side), work is underway to use predicted reinforcement along with received reinforcement.

Balakrishnan and his colleagues at the University of Missouri-Rolla, have been working on applying adaptive critic based neurocontrol for distributed parameter systems (e.g. [11], [12], [26], [36], [37], [38]). The objectives of this research are to develop and demonstrate new adaptive critic designs, and to analyze the performance of these neurocontrollers in controlling parabolic, hyperbolic, and elliptic systems.

Barto and his colleagues at the University of Massachusetts, have been working on methods to allow an agent learning through reinforcement-learning to automatically discover subgoals (e.g. [28], [40]). By creating and using subgoals, the agent is able to accelerate its learning on the current task, and to transfer its expertise to other related tasks. Discovery of subgoals is attained by examining commonalities across multiple paths to a solution. The task of finding these commonalities is cast as a multiple-instance learning problem, and the concept of diverse density is used to find the solution.

KrishnaKumar at the NASA Ames Intelligent Flight Controls Lab, and Neidhoefer, at Accurate Automation Cooperation, show an interesting implementation and application of adaptive critics ([14]). The basic idea is that if a nonlinear system can be linearized at representative points in the operational envelope, then the solution to the Ricatti equation at each point can be used as the Bellman Value function (“cost to go”) for DP. If the Ricatti solutions also show a degree of statistical correlation, then an “Immunized” scheme (which mimics the building block scheme of biological immune systems) can be used with Ricatti solutions as “building blocks” to act as HDP Immunized Adaptive Critics (IAC).

Lendaris and his colleagues at the Portland State University NW Computational Intelligence Laboratory (NWCIL) have focused the past few years on exploring application issues related to ADP, in particular, the DHP version (e.g. [18], [19], [21], [22], [23], [24], [25], [48], [49], [50], [51], [52], [53]). Much of the material reported in this chapter is an outgrowth of that work. A MATLAB based DHP computational platform has been developed, and is available for downloading and use from the NWCIL Web site: www.nwcil.pdx.edu. Key recent research and application results related to ADP involve the use of Fuzzy Logic structures for the controller, critic, and/or plant in the DHP ADP method (see Section 6.10). A recent application project is the design of a nonlinear

controller for a hypersonic-shaped aircraft known as LoFlyte[®] [20]. Current work focuses on exploring methods of J^* surface generation for fast optimal decision/control design.

Prokhorov and his colleagues at the Ford Research Laboratory have done a significant amount of work in developing stability analysis techniques for neural networks (e.g., [3], [4], [10]). An interesting application of the AC method was their experiment with a “real” ball and beam system. The benchmark ball and beam system was built in the lab, and different approaches were used to control the system. Neural networks were used in three roles: 1) to identify the system, 2) for the controller, and 3) for the critic. In one of their studies, they made the problem even more difficult by applying a sticky adhesive to the surface of the beam [9]; the ACs successfully handled the problem.

Saeks and his colleagues at Accurate Automation Corporation have been working with a variety of AC and adaptive dynamic programming implementations (e.g. [30], [45], [46]). Some of these implementations include an ADP algorithm based directly on the Hamilton-Jacobi-Bellman equation, and includes a continuous time stability proof that remains valid during on-line training. In [30], demonstrations of this algorithm are given for i) the linear case, ii) the nonlinear case using a locally quadratic approximation to the value functional, and iii) the nonlinear case using a (potentially global) radial basis function approximation of the Value function. Another AC implementation has been developed suitable for real-time applications [32], [33]. This is a time-varying linear critic methodology based on LQR theory. Applications of these algorithms have included controlling the hybrid power system of a hybrid electric vehicle, pitch control in simulated aircraft problems, simulated X-43 autoland analysis, path-planning for autonomous UAV flight, and the guidance module of a satellite formation flying algorithm.

Active work is also being performed by researchers such as Paul Werbos, Jennie Si, Sylvia Ferrari/Robert Stengel, and Ganesh Venayagourthy/Donald Wunsch. Please refer to their respective chapters in this book for a summary of their work.

6. APPLICATION ISSUES

In this section, we discuss various considerations that are important to the *application* of ADP. Before getting to the specifics, we comment that two major issues confronted in establishing practical training protocols are (1) the choice of control scenarios, and (2) the setting of values for the different parameters that govern the dynamics of the

learning process. The control scenarios aspect includes the selection of regulation points, targets/target trajectories, initial plant states, noise regimes, and reset conditions (i.e. when should a training run be terminated). Training parameters of critical importance include the learning coefficients for both the critic and the controller, and the discount factor γ used in the Bellman recursion.

6.1. Problem Formulation. The mathematical formalism used in previous sections indicates that the plant's state vector $X(t)$ is input to the critic and to the controller. An important pragmatic issue turns out being *what to include in the definition of $X(t)$* for ADP computational purposes? The control engineer using this methodology must have deep understanding of the problem context and the physical plant to be controlled to successfully make the requisite choices for $X(t)$. A strong suggestion is to invoke your engineering intuition and whatever rigorous knowledge is available to satisfy yourself that the variables you select as inputs to the controller and critic are sufficient to assure that at every point in the corresponding state space, there will exist a unique action for the controller to take. If you decide such uniqueness is questionable, then you may have to estimate some (even a hybrid) variable that will make the state space unique. If this is not achieved, all is not lost, but more complex learning structures, e.g., recurrent neural networks, may have to be used, and these are more difficult to train.

Not all mathematically describable states are observable; and even if they are in principle, there may be instrumentation constraints. Further, there are cases where we might be able to measure certain system variables (e.g., acceleration) whereas theory suggests fewer variables (e.g., only position and velocity) are required. But, experience informs us that in some situations, inclusion of the additional measurement could make the ADP process work better - e.g., if the learning device has to infer the equivalent of acceleration to satisfy certain control objectives, providing acceleration directly might be beneficial. However, "more" is not always better, as more inputs potentially add to the computational and inferencing burden. In such a case, one could explore option(s) equivalent to providing position and acceleration instead of position and velocity.

When performing the problem-formulation task, it is useful to discern whether the plant is decomposable - that is, to determine whether certain aspects of the plant dynamics may be considered to be only loosely coupled. If so, this could be useful while crafting the Utility function (discussed below), and even provides the possibility that an

equivalently loosely decoupled controller architecture might be appropriate.

While it may border on the obvious, another aspect of problem formulation that is critical to success is a clear statement of the control objectives. Only after clarity of the objectives is at hand, is one in a position to give explicit attention to the fundamental issue of how these objectives are to be represented for ADP application. The choice of this representation is a prerequisite to the next task, and is one of the key determinants of the eventual success or failure of the ADP design process.

6.2. Crafting the Utility Function. The Utility function is the only source of information the ADP process has about the task for which it is designing the controller. When the statement is made that Dynamic Programming designs an *optimal* controller, optimality is defined strictly in terms of the Utility function. It is important to recognize that a different Utility function will (typically) yield a different controller. The two key creative tasks performed by the user of ADP are

- (1) deciding what to include in the $X(t)$ vector, as discussed in the above sub-section,
- (2) *crafting* the Utility function in a manner that properly captures/embodies the problem-domain requirements, and yields a desirable controller.

One mathematical formalism suggests designating the control task in terms of a reference trajectory, say $X^*(t)$ (which could in principle be obtained from a reference model), and defining the Utility function directly as $U(t) = ||X(t) - X^*(t)||$ (e.g., see [41]). In practice, however, one finds that the ADP process can often be improved by treating some of the components of $X(t)$ in a non-uniform manner within $U(t)$. For example, different relative weightings might be used for various components, or more dramatically, some of the error components might use different powers, or alternatively, have nonlinear coefficients (e.g., see [21], [42]). Further, as suggested in [16] and [23], there is often substantial benefit to paring down $U(t)$ to contain the minimum number of terms necessary to accomplish the task (what these are, however, are not always easy to determine *a priori*).

We reproduce here a sequence of Utility functions reported in [21] that were crafted to represent an increasing set of constraints stipulated in the problem definition phase. The context was to design a steering and velocity controller for a 2-axle, terrestrial, autonomous vehicle; the task was to change lanes on a multi-lane road.

The first Utility function defined in that paper is an example of the suggestion above that it may be appropriate to have different weightings of the state variables:

$$U_1 = - \left(\frac{1}{2} y_{err} \right)^2 - \left(\frac{1}{8} v_{err} \right)^2 - \frac{1}{16} (\dot{v})^2$$

The design objectives that motivated this Utility function definition were (a) reduce distance to centerline of adjacent lane (y -error) to zero, (b) reduce velocity error to zero and (c) don't be too aggressive in making the velocity corrections.

The second Utility function is an example where a non-linear rule is incorporated. To accommodate a stipulated requirement for handling a sudden change of friction between the tire and the road (e.g., hit an ice patch), an SI (sliding index) term was crafted to provide a proxy indication of where on the nonlinear tire-model curve (of tire side force vs. tire slip angle) the vehicle was operating in:

$$SI = \left(-10 \left(\frac{\frac{\partial a_y}{\partial \alpha_f} - \left(\frac{\partial a_y}{\partial \alpha_f} \right)_{base}}{\left(\frac{\partial a_y}{\partial \alpha_f} \right)_{base}} \right) \right)$$

where $\left(\frac{\partial a_y}{\partial \alpha_f} \right)_{base}$ is the slope at the linear portion of the curves. The terms in SI are calculated via (small) applied steering inputs and measured resulting side forces generated at the tire-road interface (via a lateral accelerometer on the vehicle). So defined, the sliding index approaches a value of 10 when sliding is occurring, and approaches zero for no sliding.

Then, a new Utility function was crafted as follows:

$$U_2 = \begin{bmatrix} U_1 & \text{for } SI < 3 \\ U_1 - \frac{1}{4}(SI)^2 & \text{for } SI \geq 3 \end{bmatrix}$$

The SI value was input to the Critic and the Controller, and with this Utility function the DHP process developed a controller that successfully dealt with an ice patch in the road (and similarly, for a lateral wind gust), as described/discussed in [21].

In the third Utility function of the above reference, an additional term to limit lateral acceleration was included to accommodate a stipulation in the problem description concerning passenger "comfort" in automobiles, or for trucks, a "low tipping" requirement:

$$U_3 = U_2 - \frac{1}{8} (a_f)^2$$

The reader may consult [21] to see the performances of the sequence of controllers generated by the DHP process using the above sequence of Utility functions.

Another kind of Utility function modification is to add time-lagged values of selected state variables, to help account for known delays in the plant being controlled. An example of this was used for one of the Narendra benchmark problems [31], presented in [24]:

$$U(t) = [x_1(t+1) - x'_1(t+1)]^2 + [x_2(t+2) - x'_2(t+2)]^2$$

This Utility function did as well or better than more complex Utility functions previously reported in the literature for the same benchmark problem, and with substantially less computational overhead.

6.2.1. Decomposition of Utility Functions. If during the Problem Formulation task it is determined that the plant is (even approximately) decomposable, then there is a potential for crafting separate Utility functions for each of the resulting “chunks.”

In this case, it may be appropriate to define the overall Utility function as a sum of such component Utility functions, i.e., $U(t) = U_1(t) + \dots + U_p(t)$.

With such a formulation, a separate critic estimator could be used for each term. For HDP critics, one has

$$\begin{aligned} J(t) &= \sum_{i=0}^{\infty} \gamma^i U(t+i) \\ &= \sum_{i=0}^{\infty} \sum_{j=1}^p \gamma^i U_j(t+i) \\ &= \sum_{j=1}^p J_j(t) \end{aligned}$$

and for DHP

$$\nabla J(t) = \sum_{j=1}^p \nabla J_j(t)$$

In practice, this decomposition tends to speed up critic learning, as each sub-critic is estimating a simpler function.

In the case of multiple outputs from the controller, the controller learning process can also be simplified if the additive terms in the cost function correspond to separate modes, and the latter are dominated

by distinct control variables. For example, consider a two dimensional nonlinear system:

$$\begin{aligned}x_1 &= f_1(\mathbf{x}, u_1) \\x_2 &= f_2(\mathbf{x}, u_2)\end{aligned}$$

with primary cost function

$$U(t) = g_1(\mathbf{x}, u_1, t) + g_2(\mathbf{x}, u_2, t)$$

and secondary cost function

$$J(t) = J_1(t) + J_2(t)$$

In DHP this could be approached using two critics, each estimating $\nabla \hat{J}_1(t)$ or $\nabla \hat{J}_2(t)$ respectively. The complete gradient for controller training would be

$$\left(\frac{\partial}{\partial u_1} \hat{J}_1(t) + \frac{\partial}{\partial u_1} \hat{J}_2(t), \frac{\partial}{\partial u_2} \hat{J}_1(t) + \frac{\partial}{\partial u_2} \hat{J}_2(t) \right)$$

For initial training, the cross terms could be dropped and the following approximation used: $\left(\frac{\partial}{\partial u_1} \hat{J}_1(t), \frac{\partial}{\partial u_2} \hat{J}_2(t) \right)$. This simplifies the learning of the dominant plant dynamics and control effects. It may be useful to include a subsequent fine tuning of the controller via inclusion of the cross terms, unless the interactions are *very weak*.

See [22] for an example of decomposed utility functions for the steering and speed control of a 2-axle terrestrial vehicle. Also, see [41] for a related kind of critic decomposition, one the author calls ‘primitive adaptive critics.’

6.3. Scaling Variables. While theory does not speak to this issue directly, empirical evidence suggests that it is **eminently useful** to scale the components of $X(t)$ being fed into the controller and the critic (e.g., see [17]), such that each of the variable values are nominally in the range of ± 1 , particularly when the critic and/or controller are implemented via neural networks (this recommendation is dependent on the approximating structure used). Further, as indicated above, it is important to pay attention to the relative scaling of the component terms in the Utility function. The latter may hinge solely on engineering intuition related to the problem domain and the control specifications.

6.4. Selecting the Discount Factor. The original equation defined by Bellman that led to Equation (1) above is as follows:

$$(2) \quad J(t) = U(t) + \sum_{k=1}^{\infty} \gamma^k U(t+k)$$

We notice that the Value function $J(t)$ is given in terms of the current $U(t)$, plus the sum of all future values of $U(\cdot)$, pre-multiplied by a discount factor γ , where $0 \leq \gamma \leq 1$. At $\gamma = 0$, only the present value of U is used, ignoring all future values. At $\gamma = 1$, all future values are deemed equally important (“infinite horizon” version).

In principle, we might expect the γ value to be governed by the requirements of the original problem formulation. In applying ADP, however, an important issue is how the value of γ influences the ADP convergence process (via its role in the Bellman recursion, Equation 1). The degree to which this is felt is different for the HDP, DHP, and GDHP structures. To help inform your intuition about this, note that the critic outputs values that are used to train itself, so at early stages of the process, the component in Equation 1 contributed by the critic may be considered equivalent to ‘noise.’

For the HDP structures, those that directly estimate $J(t)$ values, appropriately selecting γ is critical for convergence of critic training. Common practice (e.g. [9]) is to start training with low γ values and then anneal them up (progressively increment them). The low γ values represent a high discount rate that cancels out the right hand term of the Bellman recursion. This results in the critic learning to approximate (just) the primary utility function $U(t)$. Progressively incrementing γ then causes the critic to learn how the primary costs accumulate through time to form the long-term (secondary) value function, $J(t)$.

For the DHP structures, those that directly produce $\nabla J(t)$, this annealing process tends to be less necessary, often allowing large γ values to be used from the very beginning. For higher dimensional problems, however, even for the DHP structure, it has been found useful to “schedule” the γ values. A reasonable rule of thumb is that even if a larger value of γ is suggested by the problem formulation and/or the ADP structure type, use a small value at the early stages of the ADP process, while the state of knowledge in the critic is low, and as the critic’s training proceeds, incrementally increase γ to higher levels.

6.5. Selecting Learning Rates. As mentioned earlier, the ADP structures addressed here all include two training loops: one for the critic and one for the controller. A *separate learning rate* (or learning-rate schedule) is associated with each training loop. In certain special cases, theory may be invoked to determine approximate desired values for the learning rates. For example, in Prokhorov et al., [57], success was reported using dual Kalman filters to automatically adjust certain parameters, though this approach adds substantial computation to the

process. More generally, however, “rule of thumb” is our primary guide, and even these are determined empirically within given problem contexts. Determining the values of these and other parameters turns out being the most labor-intensive aspect of employing the ADP methodology. In some cases, the user gets the feeling that the process requires an exhaustive search. The NWCIL DHP computational platform mentioned in Section 5 (available at www.nwcil.pdx.edu) provides a capability to experiment with lots of parameter values with minimal human intervention.

As is well known, the learning rate values mentioned above are important determinants of the training loop dynamics - and in particular, whether or not the process will converge, and if so, the convergence rate. Pragmatically, one determines useful ranges for the two learning rates empirically. Fortunately, once values are found that result in a convergent ADP process, these values are relatively robust to other process parameter changes the user may wish to explore. As with the γ values of the previous section, once useful ranges for learning rates are found, annealing (scheduling) the learning rates is also beneficial. The direction of annealing/scheduling in this case is the opposite: start with larger learning rates, and anneal downwards as learning progresses.

During the process of discovering useful values of the learning rates, if a selected set of rates results in the ADP process converging, but very slowly, then increase the learning rates incrementally - until one or both of the incremented values causes the process to diverge. Then just back down a notch or two. The more usual situation, however, is that early experiments result in a divergent process; in these cases, it is useful to observe which loop diverges first. The rate in this loop is adjusted first. Intuitively, since the controller is designed based on information acquired by the critic, it would make sense to use a larger learning-rate for the critic (to have it learn faster) than the controller. Indeed, there is empirical evidence for this. However, we have also seen counter examples, where it worked better for the controller to have a higher learning rate. The rule-of-thumb we have developed is to start with a ratio of about 10:1 for the learning rates, the critic’s being the larger one. Typical learning-rate values found useful in the problem domains explored by the first author in the past have been between 0.001 to 0.01, and sometimes up to 0.1.

Summary for Section 6.5

General summary:

Guidance for selection of ADP process parameters is via “rules of thumb”.

ADP parameter-value determination is the most labor-intensive aspect of employing the methodology.

Specific to this subsection:

Learning rate values determine training loop dynamics.

It is useful to use separate rates in each training loop.

One Rule of Thumb: Use a ratio of about 10:1 for the learning rates, the larger one for the critic loop (however, see caveat in the text).

To determine useful range(s) for learning rates, start exploration with (sometimes very) low values to find at least one set that will yield convergence; increase incrementally until process no longer converges; then back down a notch or two.

If no combination is found that yields convergence, see next subsection.

Learning-rate values found useful in applications to date (by first author and co-workers) for the critic loop are between 0.001 - 0.01, and sometimes up to 0.1.

Once useful ranges of learning rates are determined, scheduling (annealing) the rates within these ranges may be beneficial during the design runs.

Scheduling of Learning Rate values goes from large to small (in contrast to scheduling gamma values of previous sub-section, which goes from small to large).

6.6. Convergence of the ADP (controller design) process. The task of getting the ADP process to converge involves a carefully orchestrated selection of all of the above items. Experience indicates that there is strong interaction among their values in how they affect ADP convergence. If after scaling has been accomplished and exploration of learning rate and γ values has been performed with no successful convergence, we suggest reconsidering the Utility function formulation. We have examples of situations where seemingly minor changes in formulation of the Utility function resulted in dramatically different ADP convergence behavior and resulting controller design. Associated with this, it may also be useful to reconsider the selection of variables being used as inputs to the controller and to the critic (cf. discussion in Section 6.1 as well).

6.7. Designing the Training “Syllabus”. Specific attention must be given to the design of the training regimen. Many issues need to be considered. In the control context, a key issue is *persistence of excitation*, which entails a requirement that the plant be stimulated such that all important modes are excited “sufficiently often” during the

learning process. Additionally, it is also important that the full range of controller actions are experienced. A key rule-of-thumb in designing the regimen is to start the training with the simplest tasks first, and then build up the degree of difficulty. The truck backer-upper project of [35] provides an excellent example of this training principle, albeit in the context of a different learning methodology (Backpropagation through time).

The above rule-of-thumb includes considerations such as choosing initial plant states near regulation points or target states, selecting target trajectories that remain within a region of state space with homogenous dynamics, and initializing the controller with a stabilizing control law. This last approach falls under the topic of using *a priori* information to pre-structure either the controller or critic (more on this below). As the easier scenarios are successfully learned, harder scenarios are introduced in a manner that persistence of excitation across the entire desired operating region is achieved. In this stage, initial conditions for the plant are chosen farther and farther from regulation points, target trajectories are chosen so as to cross boundaries in qualitative dynamics, etc. The progression continues until the entire operating range of the controller is being exercised in the training runs.

A useful practice employed by the authors is to brainstorm how we would train animals or humans, including ourselves, to learn the given task. We then transform the insights gained into candidate training syllabi for the given ADP task.

6.8. Stopping/Reset Criteria. Another operational issue to consider is when to stop the learning process and start over again. In the well-known pole-cart problem, there is a straightforward decision: When the pole drops, stop the process, reset the pole, and continue the training process (e.g. see [16], [17]). As another example, consider training a steering controller for a 4-wheeled terrestrial vehicle to change lanes on a highway: if the vehicle goes off the road, rather than continuing the training process to see if the controller can learn to get the vehicle back on the highway, instead, stop the process as soon as the vehicle goes “out of bounds,” return to the starting point, and continue the training, starting the controller and critic weights (in the NN context) where they left off (e.g., see [21], [22]). The idea is to give an opportunity to improve the controller based on the design it had just before going out-of-bounds, rather than after it got “mired in the mud,” as it might do in attempting to get back on the highway in the steering example. This idea may easily be generalized: Specify limits for each component of $X(t)$ (and $u(t)$ if appropriate) being used in the

Utility function, create an out-of-bounds monitoring procedure, when an out-of-bound condition is detected (for one or more of the monitored variables), stop the process, return to an appropriate starting point, and continue the training.

This stop/reset strategy may also be usefully applied in those cases where the critic continues to diverge, no matter what choices are made with learning rate and/or other parameters. After a relatively “sweet” spot in the parameter values has been determined, even if the process does not converge by itself, the stop/reset strategy has been successfully employed to get the system to converge.

6.9. Simultaneous vs. Sequential Operation of Critic and Controller Training Loops. Once a forward computation is performed through the controller and plant, and a critic output is obtained (estimate of $J(t)$ or its derivatives), the ADP system is poised to perform a learning cycle in each of the two training loops. One strategy would be to simultaneously perform a learning cycle in both. This strategy works, and indeed, the authors routinely use it. However, experimentally determining values for the ADP process parameters discussed above is sometimes more difficult with this strategy than with other possibilities. In some early papers (e.g. [42], [43], [47], [57], [60], a “flip-flop” strategy was proposed wherein training was performed a number of times (called an epoch) in one loop while the training for the other loop was put on “hold”, and then during the next epoch, the roles of being trained and being on hold were flipped. This flip-flop sequencing continued until the whole ADP process converged. While this strategy tends to be easier to get to converge, its convergence rate is slower than for other alternatives. This slower convergence is a consequence of losing information in those loops that are placed on hold. Additional strategies were subsequently developed (see [16], [17]) that also make use of the principle of separate (non-simultaneous) training, but in addition provide a means of preserving all the available information, thus avoiding the penalty of longer convergence times. The mechanism for preserving the information is called “shadow critic” in the critic training loop, and “shadow controller” in the controller training loop. The shadow concept entails performing the training updates in a COPY of the critic (rather than in the critic itself) and in a COPY of the controller during their respective “hold” epochs. Then at the end of the “hold” epoch, the design in the COPY (shadow version) is uploaded to the in-line version as a starting point for training during the next epoch. Various combinations are described: Shadow Critic Only; Shadow Controller Only; Shadow Critic and Shadow Controller. The

motivating benefit for using these alternate strategies is their enhanced convergence performance. In addition, however, for some limited cases explored, the controller designs generated via the various strategies had some qualitative differences as well [24].

More recently, the Shadow Controller concept was incorporated in a proposed design of a method to deal with stability issues that arise when the ADP method is to be used in an on-line context [19]. See Section 6.11 below.

6.10. Embedding a-priori Knowledge. If *a priori* knowledge is available about the problem domain that may be translated into a starting design of the controller and/or the critic, then it behooves us to use this knowledge as a starting point for the ADP procedures. While the ADP methods may be made to converge with random initializations of the controller and critic networks (usually only applicable in off-line situations), it is generally understood that the better the starting controller design, the “easier” it will be for the ADP process to converge. Another way to look at this is that if the starting controller design is “close” to an optimal design (e.g., the human designers already did a pretty good job), then the ADP system’s task is one of *refining* a design - and this is intuitively easier than having to explore the design domain to even get to what is a *starting* point in the assumed context.

There are a variety of ways to obtain *a priori* information about the problem domain that can be used to initialize the trainable function approximator in the ADP process. For example, consider a system with an existing controller, but the controller’s design is known to be non-optimal and it is desired to improve the design. One could train a neural network to copy this controller, substitute this NN in place of the controller, and implement an ADP process to optimize the controller design. If the ADP requires a differentiable plant model, then such a model will also have to be developed before starting the process. With this starting controller design, one would begin the ADP process with a long epoch to train just the critic, and then transition into one of the strategies described in the previous section to incrementally improve both, the critic’s estimate of J^* and the corresponding controller design.

An alternate location to embed *a priori* knowledge is in the critic. For example, in the context of an LQR (linear quadratic regulator) problem, the J^* surface is known to be parabolic. While the various parameter values of the parabolic surface may not be known a priori, if the critic is pre-structured to just represent such surfaces, then ADP convergence is enhanced, e.g. see [54]. We think of this in terms of pre-biasing the critic’s ‘perception’ of the problem.

Often times, the key source of *a priori* knowledge resides in the head of a human expert. There is little available in the neural network literature that provides guidance on how to embed such a priori knowledge into a neural network starting design. On the other hand, a large literature has developed in recent decades describing theory and methods for using Fuzzy Logic to capture such human expertise, and further, for using Fuzzy Systems in the controls context. Space limitations preclude surveying that literature here; a couple of accessible suggestions to the reader are [58], [62]. It is important to point out here that certain Fuzzy structures qualify as trainable universal function approximators, and thus, should in principle be usable in ADP processes. Indeed, successful use of Fuzzy structures for both controller and/or critic roles, and in fact, for the plant's differentiable model, have been accomplished (e.g., see [50], [51], [53]). We summarize below an example of such an application (taken from [50]), to convey the thought process you would use to employ such techniques.

A Fuzzy structure known as a first-order TSK model (e.g. see [62]) offers a direct approach for representing the relevant characteristics of the plant, and for prestructuring both the controller and critic. A very simple model of the well-known cart-pole system was constructed using such a structure, and for DHP training, it was demonstrated that this model's effectiveness was comparable to the use of a full analytic model. This is especially interesting since no example-specific information (pole length or mass, cart mass, etc.) was included in the model.

The line of reasoning went as follows. First, it was noted that the six observable variables (related to pole angle θ and cart position x) constitute a coupled pair of second order systems. It can be inferred that the derivatives $\frac{\partial \theta}{\partial \theta}$, $\frac{\partial \theta}{\partial \dot{\theta}}$, $\frac{\partial \theta}{\partial \ddot{\theta}}$, $\frac{\partial \dot{\theta}}{\partial \theta}$, $\frac{\partial \dot{\theta}}{\partial \dot{\theta}}$, $\frac{\partial \dot{\theta}}{\partial \ddot{\theta}}$, $\frac{\partial x}{\partial x}$, $\frac{\partial x}{\partial \dot{x}}$, $\frac{\partial x}{\partial \ddot{x}}$, $\frac{\partial \dot{x}}{\partial x}$, $\frac{\partial \dot{x}}{\partial \dot{x}}$ and $\frac{\partial \dot{x}}{\partial \ddot{x}}$ are all always positive. This observation constitutes a partial *qualitative model* of the plant's dynamics. An additional observation is that application of a positive control force to the cart tends to increase x , \dot{x} and \ddot{x} , and decrease θ , $\dot{\theta}$ and $\ddot{\theta}$; this *a priori* knowledge allows setting $\frac{\partial x}{\partial u}$, $\frac{\partial \dot{x}}{\partial u}$, and $\frac{\partial \ddot{x}}{\partial u}$ positive, and $\frac{\partial \theta}{\partial u}$, $\frac{\partial \dot{\theta}}{\partial u}$ and $\frac{\partial \ddot{\theta}}{\partial u}$ negative. This collection of *assumptions* was defined as the Double Integrator Model (DIM). When the DIM was substituted into the baseline DHP training procedure in place of the true analytic plant model, the procedure successfully produced a controller 81 percent of the time (as compared to 99.99 percent with the analytic model).

Buoyed by this promising result, another piece of *a priori* knowledge was crafted out of the observable fact that when the pole is deflected from vertical, the force of gravity will tend to increase the angular

acceleration of the pole in the same direction as the deflection while also imparting an acceleration to the cart in the opposite direction. This resulted in a new pair of rules:

$$\text{If } \theta \neq 0 \text{ then } \frac{\partial \ddot{x}}{\partial \theta} \text{ is negative}$$

and

$$\text{If } \theta \neq 0 \text{ then } \frac{\partial \ddot{\theta}}{\partial \theta} \text{ is positive}$$

The DIM augmented with these two rules was called the Crisp Rule Double Integrator Model (CRDIM). When used in the DHP training procedure this model turned out being only 76 percent effective.

While initially disappointing, this result provided the context for an important conclusion: while the linguistic description of the plant's behavior is correct, the *crisp implementation* of the rules that were used actually detracted from the effectiveness of the CRDIM for controller training. By moving to a Fuzzy framework for the *entire model*, substantially improved results were obtained.

To keep the fuzzy implementation simple, only three linguistic *values* were used for each variable: POSITIVE, ZERO and NEGATIVE. A triangular membership function was used for the ZERO linguistic value (with end points scaled consistent with the expected range of the quantitative variable), and the membership functions for the POSITIVE and NEGATIVE values were defined so that the sum of membership values for any quantitative value sum to 1. The underlying observations included in the CRDIM were translated into fuzzy inference rules implemented using the sup-min operator for composition and the max operator for aggregation (cf. [62]). Height defuzzification was used, with centroid values of 1, 0 and -1 for POSITIVE, ZERO and NEGATIVE output values respectively. It should be clear that this Fuzzy Rule Model (FRM) would be a very poor numerical model of the system. Never the less it was 99 percent effective when used in DHP training, very close in performance to the true analytic model. 'Effectiveness' in this context is defined as the percentage of the trials in which the training procedure successfully produces a controller.

More advanced details about use of Fuzzy structures in DHP ADP systems are given in [52] and [53].

Type of Model	Effectiveness in Training
Analytic	99.99 %
Double Integrator	81 %
D. I. with Crisp Rules	76 %
Fuzzy Rules	99 %

Table 2. Effectiveness of Models in DHP Training (Pole-Cart problem)

6.11. **Stability issues.** Direct application of Dynamic Programming (DP) as originally defined would be performed off-line, and would yield an optimal controller design that would then be implemented and inserted into the object system. The DP method guarantees that the resulting controller is a stabilizing one (entailed in the definition of *optimal*).

The *approximate* DP (ADP) methods considered in this book are also intended to yield optimal controllers, albeit only *approximately* optimal ones, after the underlying iterative approximation process has converged. Once the ADP process does converge, we can assume, with reasonable theoretical justification, that the resulting controller design is a stabilizing one (e.g. see [41]).

The stability story is more complicated, however, when the ADP methods are to be used *on-line* to modify the design of the controller to accommodate changes in the problem context (i.e. to be an *adaptive* controller in the traditional controls literature sense, or, to be a *reconfigurable* controller of the more recent literature). In this (on-line) case, the question of whether the controller design is a stabilizing one has to be asked *at each iteration* of the ADP design process. This is called *step-wise* stability in [30] or *static* stability in [2].

As in Section 5, we offer here a brief review of related research (here concerning stability issues) as part of providing guidance to prospective users of the ADP.

6.11.1. *Recent Approaches to Stability Issues.* The group at Colorado State University address the issue of stability in ADP systems in terms of what they call ‘static’ stability and ‘dynamic’ stability, [2]. Their static stability means that each time the controller’s design is modified, it continues to be a stabilizing controller. This kind of stability is called step-wise stability in [30]. Their *dynamic* stability notion, on the other hand, refers to the dynamics introduced by the sequence of changed controller designs in the loop. They approach static stability with neural network controllers by first extracting the linear time-invariant

(LTI) components of the neural network and representing the remaining parts as sector-bounded nonlinear uncertainties. Integral Quadratic Constraint (IQC) analysis [1] is then used to determine the stability of the system consisting of the plant, nominal controller, and the neural network with given weight values. The dynamic stability problem is addressed by once again treating the neural network’s nonlinear components as sector bounded nonlinear uncertainties. In addition, uncertainty in the form of a slowly time-varying scalar is added to cover weight changes during learning. Finally, IQC analysis is applied to determine stability [29]. In this way, the network weight learning problem is transformed into one of network weight uncertainty; following this, a straightforward computation guarantees the stability of the network during training. The “down side” of this approach is its rather slow convergence to a design solution.

The group at the University of Massachusetts use Lyapunov methods to successfully verify qualitative properties of controller designs, such as stability, or limiting behavior, [28]. Lyapunov-based methods are used to ensure that an agent learning through reinforcement learning exhibits behavior that satisfies qualitative properties relating to goal-achievement and safety.

The group at the Ford Research Laboratories has done a significant amount of work in analyzing the stability of recurrent neural networks (RNNs), [3], [4], [9], [10]. Their work focuses on the global Lyapunov stability of multilayer perceptrons, where they assume the network weights are fixed. They perform a state space transformation to convert the original RNN equations to a form suitable for stability analysis. Then appropriate linear matrix inequalities (LMI) are solved to determine whether the system under study is globally exponentially stable. In [4], an on-line system capable of analyzing an input-output data sequence to construct a sequence of binary classifications without being provided correct class information as part of the training process is described. The system employs both supervised and unsupervised training techniques to form multiple behavior models.

The group at Portland State University’s NW Computational Intelligence Laboratory has proposed a computation/simulation approach based on the Shadow Controller concept mentioned in Section 6.9 (e.g. see [18], [19]). This approach is predicted to become viable for on-line applications as computational power continues to increase. Since the issue during on-line training of a controller is to avoid instantiating into the control loop a controller that is not stabilizing, the (DHP, in their case) training is done only on the Shadow Controller. While many issues remain to be resolved for this proposed procedure, the idea

is to determine the (local) stability of the current Shadow Controller design by performing a high-speed simulation of the closed loop (using the plant model already required for the DHP method), determine a local linearization, determine the s-plane pole locations, and from this test to determine whether the current Shadow Controller design meets minimum stability requirements; if so, upload the design to the on-line controller; if not, wait until another train/test cycle. The assumption is that stabilizing controller designs will occur sufficiently often to render the proposed procedure viable.

The group at Accurate Automation Corporation has developed an adaptive dynamic programming (incidentally, in [30], the acronym ADP is used for Adaptive Dynamic Programming, whereas in the present book ADP is used for Approximate Dynamic Programming) algorithm with a continuous time stability proof, [30]. The algorithm is initialized with a (stabilizing) Value function, and the corresponding control law is computed via the Hamilton-Jacobi-Bellman Equation (which is thus guaranteed to be a stabilizing controller for this step), and the system is run; the resultant state trajectories are kept track of and used to update the Value function in a soft computing mode. The method is repeated to convergence. In [30], this method is shown to be globally convergent, with step-wise stability, to the optimal Value function / control law pair for an (unknown) input affine system with an input quadratic performance measure (modulo the appropriate technical conditions). This algorithm has been demonstrated on the example problems mentioned in Section 5 for Saeks and his colleagues.

7. ITEMS FOR FUTURE ADP RESEARCH

As mentioned in Section 1, much progress in the development and application of Adaptive Critics has already occurred, yet much remains to be done. We comment here on two topics that appear to us to have significant potential for expanded/enhanced application of the ADP methods. One relates to employment of Fuzzy communication among the actors in an ADP system, and the other relates to speeding up the process of J^* generation in selected problem domains.

The idea for the latter is to develop a computational intelligence methodology that efficiently designs optimal controllers for *additional problems* within an assumed problem domain, *based on knowledge of existing designs in that domain*. Research on this approach is just starting at the NWCIL, but promises to benefit from broader involvement. As currently envisioned, the key ingredient of this methodology will

be a J^* Surface Generator (J^*SG). Fundamental tasks in this quest involve representation, representation, representation.

The idea for including Fuzzy communication among the actors in an ADP system (in contrast to within the various actors, as discussed in Section 6.10) is motivated by observing the process wherein, for example, a human athlete refines his/her performance based on verbal hints/instructions provided by an experienced coach (this may apply to many human activities, such as dancing, art, etc.). A potentially key location for receiving/embedding such knowledge communicated via a Fuzzy representation could be the Utility function. A hint that the Utility function could be the heart of such a refinement process may reside for us in the sequence of additions to the Utility functions described in Section 6.2, and the corresponding refinements in controller performance achieved. Each of us has many personal experiences of using verbally communicated guidance to enhance some kind of performance, and this could provide a rich source of intuition for such an approach. We encourage dialogue to begin within our research community.

REFERENCES

- [1] C. Anderson. Approximating a policy can be easier than approximating a value function. Technical Report CS-00-101, Colorado State University, 2000.
- [2] C. Anderson, R. M. Kretchner, P. M. Young, and D. C. Hittle. Robust reinforcement learning control with static and dynamic stability. *International Journal of Robust and Nonlinear Control*, 11, 2001.
- [3] N. Barabanov and D. Prokhorov. Stability analysis of discrete-time recurrent neural networks. *IEEE Trans. On Neural Networks*, March, 2002.
- [4] N. Barabanov and D. Prokhorov. Two alternative stability criteria for discrete-time rmlp. Las Vegas, NV, Dec. 2002. Control and Decision Conference.
- [5] A. G. Barto. *Handbook of Intelligent Control*, chapter Reinforcement Learning and Adaptive Critic Methods, pages 469–491. New York: Van Nostrand-Reinhold, 1992.
- [6] A. G. Barto and R. S. Sutton. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [7] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [8] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- [9] P. Eaton, D. Prokhorov, and D. Wunsch. Neurocontroller for fuzzy ball-and-beam systems with nonlinear, nonuniform friction. *IEEE Trans. On Neural Networks*, pages 423–435, March 2000.
- [10] L. Feldkamp, T. Feldkamp, and D. Prokhorov. *Intelligent Signal Processing*, chapter An Approach to Adaptive Classification. IEEE Press, 2001.
- [11] Z. Huang and S. N. Balakrishnan. Robust adaptive critic based neurocontrollers for missiles with model uncertainties. *2001 AAA Guidance, Navigation and Control Conference, Montreal, Canada*, Aug 2001.

- [12] Z. Huang and S.N. Balakrishnan. Robust adaptive critic based neurocontrollers for systems with input uncertainties. *Proceedings of IJCNN'2000, Como, Italy*, pages B-263, July 2000.
- [13] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237-285, 1996.
- [14] K. KrishnaKumar and J. Neidhoefer. Immunized adaptive critics. invited session on Adaptive Critics, ICNN '97, Houston, 1997. A version of this was presented at ANNIE '96, November 10- 13, St. Louis, MO.
- [15] K. KrishnaKumar and J. Neidhoefer. *Immunized Adaptive Critic for an Autonomous Aircraft Control Application*. Artificial immune systems and their applications. Springer-Verlag, Inc., 1998.
- [16] G. G. Lendaris and T. T. Shannon C. Paintz. More on training strategies for critic and action neural networks in dual heuristic programming method (invited paper). *Proceedings of Systems Man and Cybernetics Society International Conference'97, Orlando, IEEE Press*, October 1997.
- [17] G. G. Lendaris and C. Paintz. Training strategies for critic and action neural networks in dual heuristic programming method. *Proceedings of International Conference on Neural Networks'97 (ICNN'97), Houston*, July 1997.
- [18] G. G. Lendaris, R. A. Santiago, and M. S. Carroll. Dual heuristic programming for fuzzy control. *Proceedings of IFSA / NAFIPS Conference, Vancouver, B.C.*, July 2002.
- [19] G. G. Lendaris, R. A. Santiago, and M. S. Carroll. Proposed framework for applying adaptive critics in real-time realm. *Proceedings of International Conference on Neural Networks'02 (IJCNN' 2002), Hawaii*, May 2002.
- [20] G. G. Lendaris, R. A. Santiago, J. McCarthy, and M. S. Carroll. Controller design via adaptive critic and model reference methods. *Proceedings of International Conference on Neural Networks'03 (IJCNN' 2003), Portland*, July 2003.
- [21] G. G. Lendaris and L. J. Schultz. Controller design (from scratch) using approximate dynamic programming. *Proceedings of IEEE International Symposium on Intelligent Control '2000, (IEEE-ISIC'2000), Patras, Greece*, July 2000.
- [22] G. G. Lendaris, L. J. Schultz, and T. T. Shannon. Adaptive critic design for intelligent steering and speed control of a 2-axle vehicle. *Proceedings of International Conference on Neural Networks'00 (IJCNN'2000) Italy*, Jan 2000.
- [23] G. G. Lendaris and T. T. Shannon. Application considerations for the dhp methodology. *Proceedings of the International Joint Conference on Neural Networks'98 (IJCNN'98), Anchorage, IEEE Press*, May 1998.
- [24] G. G. Lendaris, T. T. Shannon, and A. Rustan. A comparison of training algorithms for dhp adaptive critic neuro-control. *Proceedings of International Conference on Neural Networks'99 (IJCNN'99), Washington,DC*, 1999.
- [25] G. G. Lendaris, T. T. Shannon, L. J. Schultz, S. Hutsell, and A. Rogers. Dual heuristic programming for fuzzy control. *Proceedings of IFSA / NAFIPS Conference, Vancouver, B.C.*, July 2001.
- [26] X. Liu and S. N. Balakrishnan. Convergence analysis of adaptive critic based neural networks. *Proceedings of 2000 American Control Conference, Chicago, IL*, June 2000.
- [27] R. Luus. *Iterative Dynamic Programming*. CRC Press, 2000.

- [28] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. *Proceedings of the 18th International Conference on Machine Learning*, pages 361–368, 2001.
- [29] A. Megretski and A. Rantzer. System analysis via integral quadratic constraints: Part ii. Technical Report ISRN LUTFD2/TFRT-7559-SE, Lund Institute of Technology, September, June 1997.
- [30] J. J. Murray, C. Cox, G.G. Lendaris, and R. Saeks. Adaptive dynamic programming. *IEEE TRANSACTIONS on SYSTEMS, MAN, and CYBERNETICS, PART C: Applications and Reviews*, 32, No.2:140–153, May 2002.
- [31] K. S. Narendra and S. Mukhopadhyay. Adaptive control of nonlinear multi-variable systems using neural networks. *Neural Networks*, 7(5):737–752, 1994.
- [32] J. C. Neidhoefer. Report. Technical Report AAC-01-055, Accurate Automation Corp, 2001.
- [33] J. C. Neidhoefer. Report. Technical Report AAC-02-016, Accurate Automation Corp, 2002.
- [34] J. C. Neidhoefer and K. Krishnakumar. Intelligent control for autonomous aircraft missions. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, January 2001.
- [35] D. Nguyen and B. Widrow. *Neural Networks for Control*, chapter The Truck Backer-Upper: an Example of Self Learning in Neural Networks. MIT Press, 1957.
- [36] R. Padhi and S. N. Balakrishnan. Adaptive critic based optimal control for distributed parameter systems. *Proceedings International Conference on Information, Communication and Signal Processing*, December 1999.
- [37] R. Padhi and S. N. Balakrishnan. A systematic synthesis of optimal process control with neural networks. *Proceedings American Control Conference, Washington, D.C.*, June 2001.
- [38] R. Padhi, S. N. Balakrishnan, and T. Randolph. Adaptive critic based optimal neuro control synthesis for distributed parameter systems. *Automatica*, 37:1223–1234, 2001.
- [39] J. C. Pearce. *The Biology of Transcendence*. Park Street Press, 2002.
- [40] T. J. Perkins and A. G. Barto. Lyapunov design for safe reinforcement learning. AAAI Spring Symposium on Safe Learning Agents.
- [41] D. Prokhorov. *Adaptive Critic Designs and their Application*. PhD thesis, Texas Tech University, 1997. Department of Electrical Engineering.
- [42] D. Prokhorov, R. Santiago, and D. Wunsch. Adaptive critic designs: A case study for neurocontrol. *Neural Networks*, 8:1367–1372, 1995.
- [43] D. Prokhorov and D. Wunsch. Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8(5):997–1007, 1997.
- [44] A. Rogers, T. T. Shannon, and G. G. Lendaris. A comparison of dhp based antecedent parameter tuning strategies for fuzzy control. *Proceedings of IFSA/NAFIPS Conference, Vancouver B.C., July*, 2001.
- [45] R. Saeks, C. Cox, J. Neidhoefer, and D. Escher. Adaptive critic control of the power train in a hybrid electric vehicle. *Proceedings SMCia Workshop*, 1999.
- [46] R. Saeks, C. Cox, J. Neidhoefer, P. Mays, and J. Murray. Adaptive critic control of a hybrid electric vehicle. *IEEE Transactions on Intelligent Transportation Systems*, 3, No.4, December, 2002.

- [47] R. Santiago and P. Werbos. New progress towards truly brain-like intelligent control. *PROC WCNN '94*, pp. 1-2to1-33, Erlbaum, 1994.
- [48] L. J. Schultz, T. T. Shannon, and G. G. Lendaris. Using dhp adaptive critic methods to tune a fuzzy automobile steering controller. *Proceedings of IFSA/NAFIPS Conference, Vancouver, B.C., July, 2001*.
- [49] T. T. Shannon. Partial , noisy and qualitative models for adaptive critic based neuro-control. *Proceedings of International Conference on Neural Networks'99 (IJCNN'99), Washington, D.C., July, 1999*.
- [50] T. T. Shannon and G. G. Lendaris. Qualitative models for adaptive critic neurocontrol. *Proceedings of IEEE SMC'99 Conference, Tokyo, June, 1999*.
- [51] T. T. Shannon and G. G. Lendaris. Adaptive critic based approximate dynamic programming for tuning fuzzy controllers. *Proceedings of IEEE-FUZZ 2000, IEEE Press, 2000*.
- [52] T. T. Shannon and G. G. Lendaris. A new hybrid critic-training method for approximate dynamic programming. *Proceedings of International Society for the System Sciences, ISSS'2000, Toronto, August, 2000*.
- [53] T. T. Shannon and G. G. Lendaris. Adaptive critic based design of a fuzzy motor speed controller. *Proceedings of ISIC2001, Mexico City, Mexico, September, 2001*.
- [54] T. T. Shannon, R. A. Santiago, and G. G. Lendaris. Accelerated critic learning in approximate dynamic programming via value templates and perceptual learning. *Proceedings of IJCNN'03, Portland, July, 2003*.
- [55] S. Shervais and T. T. Shannon. Adaptive critic based adaptation of a fuzzy policy manager for a logistic system. *Proceedings of IFSA /NAFIPS Conference, Vancouver, B.C., July, 2001*.
- [56] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. *Proceedings of the American Control Conference, Boston, MA, pp. 2143-2146, 1991*.
- [57] N. Visnevski and D. Prokhorov. Control of a nonlinear multivariable system with adaptive critic designs. *Proceedings of Artificial Neural Networks in Engineering (ANNIE), ASME Press, New York, 6:559-565, 1996*.
- [58] L. X. Wang. *A Course in Fuzzy Systems and Control*. Prentice Hall, 1997.
- [59] P. J. Werbos. *Neural Networks for Control*, chapter A Menu of Designs for Reinforcement Learning Over Time, pages 67-95. MIT Press, Cambridge, MA, 1990.
- [60] P. J. Werbos. *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, chapter Approximate Dynamic Programming for Real-Time Control and Neural Modeling, pages 493-525. Van Nostrand Reinhold, New York, 1994.
- [61] B. Widrow, N. Gupta, and S. Maitra. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man and Cybernetics*, 3(5):455-465, 1973.
- [62] J. Yen and R. Langari. *Fuzzy Logic: Intelligence, Control and Information*. Prentice Hall, 1999.