A Retrospective on Adaptive Dynamic Programming for Control

George G. Lendaris

Abstract—Some three decades ago, certain computational intelligence methods of reinforcement learning were recognized as implementing an approximation of Bellman’s Dynamic Programming method, which is known in the controls community as an important tool for designing optimal control policies for nonlinear plants and sequential decision making. Significant theoretical and practical developments have occurred within this arena, mostly in the past decade, with the methodology now usually referred to as Adaptive Dynamic Programming (ADP). The objective of this paper is to provide a retrospective of selected threads of such developments. In addition, a commentary is offered concerning present status of ADP, and threads for future research and development within the controls field are suggested.

I. HISTORICAL BACKGROUND

While existence of control devices dates back to antiquity (call it Phase 1 for the controls field), it may be said Maxwell’s use of differential equations to analyze the dynamics of a flyball governor (ca. 1870) [34] that had been invented by James Watt in 1788 [65] ushered in a new phase for the controls field (call it PHASE 2). Mathematics has played a fundamental role in Phase 2, progressing through Fourier and Laplace transforms, state space methods, stochastic methods, Hilbert space methods, and more recently, algebraic and geometric topological methods. The advent of modern computers with their fantastic evolution the past few decades has also been significant, not only from the implementation point of view, but also as a driver and motivator for various mathematical and algorithmic developments as well.

An important aspect of the Phase 2 methods, however, is that the controller so designed is placed in service with no associated mechanism for modifying its design in response to context changes, be they in the plant or its environment. This phase includes at least the following well known design methods: Classical Control, Modern Control, Optimal Control, Stochastic Control, and Robust Control (e.g., [8][13][14][34][35][39][44][49][73]). Even though the progression of these methods has employed ever more sophisticated mathematical tools and insights, and yielded enhanced controllers (with respect to the criteria defined for the objectives of each approach), nevertheless, in the end, after the controller is designed, it is placed in service with no associated mechanism to modify its design in response to changes in context. To somewhat accommodate the latter, the designs in this category are often crafted to have low sensitivity (“robustness”) to select changes in plant or environment parameter values. While these controllers accommodate certain context changes, this is accomplished by virtue of margins in controller design rather than by on-line changes in the design itself.

The Phase 2 Optimal Control methods have achieved great success for linear systems (via Riccati type equations), but the success is mediated for some applications, because the methods require knowledge of plant dynamics, and not readily amenable to on-line application. The results for non-linear systems are more measured, as the available solution methods (e.g., via Hamilton-Jacobi type equations), which also require knowledge of plant dynamics, are not as generally applicable.

In a number of applications, the context changes so much during operation that the fixed controller designs resulting from the above methods were not sufficient. A design path emerged that accommodates context variations via on-line instantiation of different controller designs based on the observed variations, ushering in another new phase in the development of the controls field (call it PHASE 3).

A large segment of the methods developed for this Phase 3 may be labeled PARTITIONING, as they partition a nonlinear operating region into approximately linear regions, and develop a linear controller for each. These methods may be said to focus on changes in the environment component of the context in which the control system operates. The various methods have different means of “knowing” which context is the current one. In general, once the specific current context is known, a previously designated controller or controller design process is then instantiated. Partitioning methods have appeared in a variety of other technology sectors as well, e.g. artificial intelligence, neural networks, Fuzzy logic, statistics, etc. Thus, the partitioning methods have appeared under a variety of labels – e.g., multiple models, piecewise models, mixture of experts, Fuzzy models, local regression, etc.

Another class of methods developed for Phase 3 may be labeled ADAPTIVE CONTROL and LEARNING CONTROL. Both operate over a specified pool of controllers, and they converge on a design within this pool based on a sequence of state and/or environment observations and/or performance evaluations. In the Adaptive Controls case, the engineer specifies a set of available controllers (via parameterized models) and an algorithm to select from this set based on observations. In the Learning Controls case, the engineer specifies a parameterized controller structure, and a corresponding algorithm to incrementally adjust the parameters to converge on an appropriate design as each new situation is encountered.

While Adaptive Control methods have been developed that generate controller designs with guaranteed performance (for the context in which they are designed), the de-
signs typically do not enjoy the quality of being “optimal” (relative to user-specified criteria).

Of significance here is that the Adaptive and Learning Control methods above do not retain memory of solutions as they are achieved. The two methods differ primarily in the amount of a priori information embedded in their respective pool of controllers, and the off-line vs. on-line aspects. Apropos the latter, the Adaptive Control methods normally provide a guarantee that switching among the policies in the set can be done safely, say related to stability, in an on-line manner [3][39][56]. Historically, such guarantees were not available with Reinforcement Learning methods (the methods being focused on in the present paper) but extensions demonstrate selection methods that permit on-line operation as well (e.g., [1][2][16][28][29][32][40][43]).

In a recent paper [31], I posited a next phase for development of the controls field (calling it PHASE 4), and dubbed it Experience Based Identification & Control. The focus defined for this new phase relates to the quest of many researchers in our field(s) to have our technology achieve more human-like capabilities for identification and control. Human performance levels for such tasks clearly depend on effective and efficient use of experiential knowledge.

While the controls field has indeed accumulated remarkable achievements, even exceeding human control capabilities for some applications, still, substantial additional progress is needed toward building into machines the ability to employ experiential knowledge (experience) when performing system identification and when coming up with a good previously designed controller for a given situation, and importantly, to do so effectively and efficiently.

The Experience Based (EB) ideas of that paper [31] are motivated via two key observations of human abilities:

1. After a human learns a set of related identification and/or control tasks, when presented with a novel task of the same genre, the human generates reasonably optimal performance on the new task (i.e., effective selection from experience).
2. The more knowledge a human attains, the speed and efficiency of performing tasks are improved. [In contrast, for AI systems thus far developed, the more knowledge acquired (typically stored as “rules”) the slower the decision/action processing.]

The EB formulation entails a memory of previously developed solutions to a control task for a given context (called experience), and as context changes, entails selecting among the solutions in the experience repository (the applicable memory), and doing so in an effective and efficient manner (cf. above list). Fundamental to the posited Phase 4 methods will be selection strategies (of solutions existing in the experience repository), to be designed by optimization methodologies for effectiveness and efficiency. A candidate optimization method is a “higher level” application of ADP [31].

While there is considerable conceptual overlap between the Phase 3 methods and those posited for Phase 4, nevertheless, it is claimed there are sufficient distinctions among the respective guiding principles that explication and pursuit of the EB ideas are warranted, as some of the distinctions seem crucial to the scalability issues (e.g., speed of selection vs. size of the repository/memory) to be faced when attempting to develop human-level performance by a computing agent.

Further, whereas multiple-model methods have historically used linear models (except more recent neural network ones [49]), no such constraint is involved in the proposed EB method.

II. WHENCE ADAPTIVE DYNAMIC PROGRAMMING FOR CONTROL APPLICATIONS?

As the Phase 2 methods evolved, a key focus in the controls field became one of achieving optimal control, beginning with linear systems, and progressing to selected nonlinear applications. A number of difficult issues relating to this task were addressed and solved, entailing successful development of elegant mathematical concepts and algorithms. One of these was Bellman’s Dynamic Programming method [6], and the subsequent Hamilton-Jacobi-Bellman (HJB) equation for solving a broad class on non-linear optimization problems. However, a nagging issue remained: the high computational cost associated with carrying out these formulations, plus a difficulty with applying them as on-line procedures (e.g., the backward-in-time aspect of Dynamic Programming).

Another focus of Phase 2 was on maintaining desired system performance in the face of changes in plant parameter values and/or unknown disturbances from the plant’s environment. The H∞ methods were developed during this phase, and in addition, there evolved the methods of partitioning, adapting, and learning that became the next phase (Phase 3).

In the meantime, the field of neural networks (NNs) was also evolving, and notice was made by researchers that the innate capability of NNs to implement non-linear mappings and their ability to serve as universal function approximators [12] were attributes of importance for potential application to controls. An early (if not the first) proposal to employ NNs in the controls context was made by Werbos [67], and important early contributions were made by Narendra [41]. Descriptions of early applications are contained in [38], and descriptions of design issues associated with employing NNs for control are included in [71]. A more recent set of example successful NN applications to controls appear in [42].

During the current decade, significant work has occurred in the NN field relating to application of NNs to the Phase 3 methods, and to application of NNs to the Phase 2 control optimization task. The latter has been via a method that employs Adaptive Critics (see next section) to perform a process that approximates the Dynamic Programming method. In recent years, this approach has been dubbed ADP, for Adaptive Dynamic Programming (or for Approximate Dynamic Programming). This evolved from the learning approach called Reinforcement Learning.

ADP is now poised to make simultaneous contributions to issues from both Phase 2 and Phase 3 methods, that is, to yield optimal and adaptive feedback control structures via learning. ADP is also poised to make an entry to the proposed Phase 4 methods.

III. LEARNING VIA NEURAL NETWORKS

Neural Networks extract problem-domain knowledge by interacting with their environment using various strategies, typically referred to as learning algorithms. These
algorithms fall into three general categories: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. The Supervised Learning category entails a “teacher” role having at its disposal full knowledge of the problem context (about which the “pupil” NN is to learn), and in particular, has available a collection of input/output data pairs with which to conduct the pupil’s learning process. In the Unsupervised Learning category, the input-output notion does not apply; rather, there is simply a batch of data available from the environment, and the pupil NN’s task is to “discover” attributes of the data, which subsequently can be used by the human users for a variety of purposes. Whereas in Supervised Learning the teacher role has full knowledge of the output values desired from the pupil NN in response to each given input, in Reinforcement Learning, the teacher does not know the detailed actions the NN should be performing, and can only provide qualitative, sometimes infrequent, feedback about its performance. For example, when we were infants learning to walk, there was no teacher to provide data to us regarding how each of our muscles should have been functioning to accomplish each small movement; rather, only after we took a step and fell down were we provided the general assessment that we ‘fell down’ (plus some associated information about the form of our falling down). As our walking improved, the kind of feedback changed (e.g., to “wobbliness”, “clumsiness”, etc.). In the end, based on these ‘reinforcement signals,’ we learned how to walk. The study of reinforcement learning in our field goes back at least to the early 1980s [5].

A class of Reinforcement Learning methods that employs computational entities to critique the actions of other such entities is known as Adaptive Critics. The ‘critiquing’ in the adaptive critic methodology normally takes place over time, and a problem context which turns out being a natural for adaptive critic methodology normally takes place over time, entities is known as Adaptive Critics. The ‘critiquing’ in the computational entities to critique the actions of other such entities is known as Adaptive Critics. The ‘critiquing’ in the adaptive critic methodology normally takes place over time, and a problem context which turns out being a natural for adaptive critic methodology normally takes place over time, entities is known as Adaptive Critics.

In the end, based on these ‘reinforcement signals,’ we learned how to walk. The study of reinforcement learning in our field goes back at least to the early 1980s [5].

A class of Reinforcement Learning methods that employs computational entities to critique the actions of other such entities is known as Adaptive Critics. The ‘critiquing’ in the adaptive critic methodology normally takes place over time, and a problem context which turns out being a natural for application of the Adaptive Critic method is that of controls, wherein it is desired to design a controller for a ‘plant’ based on specified design criteria, typically involving performance over time. The critic/reinforcement method paved the way for evolving learning approaches to solving controller design problems in the context where little or no a priori knowledge of needed controller actions is available, and/or a context wherein substantive changes occur in the plant and/or environment that need to be accommodated by the (learning) controller. The term ‘adaptive critic’ was applied in the machine learning context back in 1973; in the early stages of this evolution, it was still required to endow the teacher role with some knowledge of the control actions needed [70]. Continuing up into the early 1980’s, AI researchers who employed critics were typically obliged to provide the critic with domain-specific knowledge [11]. In 1983, however, a very important extension was made in which the critic also learns, in this case, to provide increasingly better evaluation feedback; this removed the requirement for providing the teacher a priori knowledge about desired controller actions [5]. Historically, Widrow used the term ‘adaptive critic’ to imply learning with a critic. Had just the term ‘critic’ been used by Widrow (and others), the adjective ‘adaptive’ could naturally have been applied to create the term ‘adaptive critic’ after the learning capability was added to the critic by Barto, et al. Historical precedence notwithstanding, current usage employs the term ‘adaptive’ to refer to the critic’s learning attribute (dating back to [68]).

In their initial work, Barto, et al, cited the checkers-playing program written by Samuel [55] as an important precursor to their development; they characterized Samuels program as implementing a method “...by which the system improves its own internal critic by a learning process.” The result here was a system in which two learning processes are taking place: one for the critic and one for the controller NN. The Barto, et al, context was one of designing a controller with the (not necessarily modest) objective of providing stable control. In recent years, a more aggressive objective has been to use the Adaptive Critic methodology to accomplish design of optimal controllers, with stability and robustness included [32].

Application of the Adaptive Critic methodology via NNs to the design of optimal controllers benefited from the confluence of three developments: 1) Reinforcement Learning, mentioned above, 2) Dynamic Programming, and 3) Backpropagation-based learning algorithms.

IV. OPTIMAL CONTROL

The quest to design controllers that are best has been with us for some time. When attempting to make something best (“optimize”), a fundamental concept is that of a criterion function – a statement of the criteria by which ‘best’ or ‘optimum’ is to be determined. Criterion functions go by various names, e.g., ‘cost function’ and ‘utility function’.

Entire books have been written to describe various approaches, results, and methods for designing optimal controllers (e.g., [4][9]). Applications range from designing controllers for linear systems based on quadratic criterion functions, to the more complex non-linear systems. For the linear-quadratic case, complete solutions are available (Ricatti Equation, etc.) and are computationally tractable; for the general non-linear case, while the method known as Dynamic Programming [6][7][15] is a unified mathematical formalism for developing optimal controls, historically, its application has not been computationally tractable. Also, since Dynamic Programming is a backward-in-time method, it is not directly suitable for on-line applications. The good news in the past 15+ years is that with Adaptive Critic methods, a good approximation to the Dynamic Programming method can be implemented for designing a controller in a system with full non-linear plant capability – and is implementable in a forward-in-time and a computationally tractable manner.

V. DYNAMIC PROGRAMMING

Dynamic Programming (DP) remains the only general approach for sequential optimization applicable under very broad conditions, including uncertainty [7]; its application to dynamical systems has entailed discrete-time treatments. While the details of carrying out the Dynamic Programming
method are complex, the key ideas underlying it are straight forward. The method rests on Bellman’s Principle of Optimality, which stipulates that when developing a sequence of control actions (a “control policy”) to accomplish a certain trajectory, an optimal trajectory has the property that no matter how an intermediate point is reached, the rest of the trajectory (from the intermediate point to the end) must also be optimal. This turns out being a powerful principle to apply in a critical part of the reasoning path toward attaining the optimal solution.

A necessary early step in designing a controller is to formulate a (primary) utility function, $U(t)$, that embodies the design requirements for the controlled system and problem context. This step must be done with care, as this is the only source of “information” available to the (automated) design process for the objectives of the controller it is to design, and further, because the resulting controller’s attributes and quality of performance are intimately influenced by this utility function. This function provides the ‘cost’ incurred while transitioning from a given state to the next one.

A secondary utility function is used to perform the optimization process, and is defined in terms of the Utility function as follows: $J(t) = \sum_{i=0}^{\infty} \gamma^{i} U(t + k)$, $0 < \gamma \leq 1$. $J(t)$ is known as the Value function (also called a Cost-to-Go function). [Bellman used the technically correct terminology Value functional, but much of our literature uses function instead; the latter is used in this paper.] 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.

Unfortunately, the calculations required for DP 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 [37]) – and now, ADP is an approximating method that employs Adaptive Critic type of Reinforcement Learning, with an important advantage of being a forward-in-time procedure (useful for on-line application).

We leave this subsection with the reminder that the Dynamic Programming process is intended to come up with a sequence of controls that is optimal in a stipulated sense, yielding what is called an ‘optimal control policy’, i.e., it is a process of designing an optimal controller.

VI. BACKPROPAGATION

A key component of the Adaptive Critic method(s) is an algorithm known as Backpropagation. This algorithm is no doubt the most widely used algorithm in the context of Supervised Learning of feed-forward neural networks. Backpropagation by itself, however, is NOT a training algorithm, per se. Rather, it is an efficient and exact method for calculating derivatives. While this attribute of Backpropagation is indeed used as the backbone of an algorithm for supervised learning in the NN context, for the purposes of this paper, more important is the general view of what Backpropagation is, namely, it is (ingeniously) an implementation of the chain-rule of taking derivatives. The order in which associated operations are performed is important, and this prompted its inventor [66] to use the term ‘ordered derivatives’ for this context.

VII. ADAPTIVE CRITICS TO ADP

The Adaptive Critic (AC) concept is essentially a combination of Reinforcement Learning (RL) and Dynamic Programming (DP) ideas. Importantly, 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. Accordingly, AC methods were referred to as implementing Approximate Dynamic Programming (ADP), and more recently as Adaptive Dynamic Programming (also ADP). A key attribute of AC that enables on-line application is that it effectively solves the HJ optimization equations forward in time vs. the backward-in-time approach of basic DP [68].

A family of ADP structures was proposed by Werbos in the early 1990’s [68], [69], and has been widely used by others [16], [17], [18], [19], [20], [23], [24], [25], [26], [27], [28], [36], [45], [46], [47], [51], [53], [54]. 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; examples may be found in [27], [52], [57], [61], [63]. Werbos’ family (also called “ladder”) 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 [51], called Adaptive Critic Designs there.

The different ADP structures can be distinguished along three dimensions: 1) Critic Inputs; 2) Critic Outputs; and 3) Loops requiring model of plant in the training process.

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).

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 $J(t)$ (denoted $\hat{J}(t)$); and in the GDHP, it approximates both, $J(t)$ and its gradient $\hat{J}(t)$.

Plant Model Requirements: While there exist formulations that require only one training loop (e.g. [40]), 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 AC process is a gradient-based one and thus requires estimates of various derivatives within the two training loops, in an appropriately ordered fashion. Since Backpropagation is an implementation of the chain-rule for taking derivatives (cf. Section VI), it becomes an essential component for ADPs that employ NNs.

A. Model Use in Training Loops.

The base components of the ADP process are the “action” NN (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 controller and critic training.

This data is needed in the calculations for performing the controller and critic training.

ADP training is based on the Bellman Recursion:

\[
J(t) = U(t) + \gamma J(t+1), \quad 0 < \gamma \leq 1. 
\]

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 \( t+1 \).

The following is a verbal “walk through” of a few of the AC structures, pointing out why and in which loop(s) 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

\[
\frac{\partial J(t)}{\partial u_i(t)} = \sum_{j} \frac{\partial J(t)}{\partial X_j(t)} \frac{\partial X_j(t)}{\partial u_i(t)}. 
\]

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, \( \frac{\partial J(t)}{\partial u_i(t)} \), 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:

\[
\lambda_i(t) = \frac{\partial U(t)}{\partial X_i(t)} + \frac{\partial U(t)}{\partial u_i(t)} \frac{\partial u_i(t)}{\partial X_i(t)} + \lambda_i(t+1) \frac{\partial X_i(t+1)}{\partial X_i(t)} \frac{\partial u_i(t)}{\partial \lambda_i(t)} 
\]

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., \( \frac{\partial X_i(t+1)}{\partial X_i(t)} \) and \( \frac{\partial \lambda_i(t)}{\partial \lambda_i(t)} \). 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. (While some view this model dependence to be an unnecessary “expense,” it is held here that the expense is in many contexts more than compensated for by the additional information available to the learning/optimization process. Further motivation for pursuing model-dependent versions may be based on the biological exemplar: some explanations of the human brain developmental/learning process invoke the notion of ‘model imperative’ [48].)

Analysis of the remaining ADP types is similar. Results are tabulated in Table 1.

---

**Table 1. Summary of Requirements for Model in Training Loops.**

<table>
<thead>
<tr>
<th>ADP STRUCTURE</th>
<th>CRITIC</th>
<th>CONTROLLER</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDP</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ADHDP</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DHP</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ADHDP</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

---

VIII. CLOSED LOOP STABILITY ISSUES FOR ADP

To be acceptable in the field, application of any methodology for designing feedback controllers on-line must meet stringent requirements for maintaining closed-loop stability during the entire design process. It does little good to have an on-line method that yields, say, an optimal controller if in the design process the system went unstable and destroyed the plant to be controlled.

Once the ADP process does converge, we can assume with reasonable theoretical justification that the resulting controller design is a stabilizing one. 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 in later literature). In the on-line case, the question of whether the current design for the controller is a stabilizing one has to be asked of the ADP design process at each iteration. This is called step-wise stability in [40] or static stability in [1].

In early applications of ADP to the controller-design task, assuring closed-loop stability for the duration of the ADP...
process was problematic, and thus reluctance to apply the method (actually, any learning method) on-line.

A number of approaches were undertaken to deal with this issue, and various strategies for performing ADP-based procedures were developed that assured stability during the design process. Summaries of these approaches have been published, for example one in which I participated [30], but the most complete and mathematically sophisticated one(s) to my awareness are by the chair of this special session, Frank Lewis [32], whose group has contributed heavily to the field along this line. The reader is directed to such references.

I take opportunity here to mention an approach I was intimately involved with, reported in [40]. The method described there is shown to be globally convergent, with stepwise 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).

At a more general level, a good source for the status of ADP (up through 2003) is contained in the compilation of papers in [76], and more recent ones in [77].

IX. PRAGMATIC ASPECTS OF EMPLOYING ADP

As for any application domain, one begins the process with a good problem definition – in the present case, with a clear description of the plant to be controlled and its environment, and of a well-formulated Utility function that captures all the user design objectives. As noted earlier, 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 $X(t)$. 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.

When performing the problem-formulation task, it is useful to discern whether the plant is (even approximately) decomposable – that is, to determine whether certain aspects of the plant dynamics may be considered to be only loosely coupled. If so, 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 [30], i.e., $U(t) = U_1(t) + \ldots + U_n(t)$. With such a formulation, a separate critic estimator could be used for each term. In practice, this decomposition tends to speed up critic learning, as each sub-critic is estimating a simpler function. Further, such decomposition provides the possibility an equivalent loosely decoupled controller architecture might be appropriate.

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.

Keep in mind that the Utility function is the only source of information the ADP process has about the task for which it is designing the controller. Further, 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, and (2) crafting the Utility function in a manner that properly captures/embodies the problem-domain requirements, and yields a desirable controller.

In addition to the above, application of ADP has certain aspects that are as much an “art form” as anything else, especially the choices for certain ADP process parameters, such as values for $\gamma$ and for learning rate(s). Be aware that ADP parameter-value determination is the most labor-intensive aspect of employing the methodology.

Design of the training regimen requires specific attention. 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 ranges 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. This 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 (partition) either the controller or critic (cf. [21]). As the easier scenarios are successfully learned, more difficult 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 author 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. 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 successfully 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 demonstrably easier than having to explore the design domain to even get to what is a starting point in the assumed context. 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 (related pre-structuring work may be found in [74]).

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 uses 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 [58], [59], [61]).

X. WHERE DO WE GO FROM HERE?

In general, it may be said that to date, ADP methods are reasonably well along their way in the controls arena for systems in which actions are taken at discrete time steps (w states and control values may be continuous). The call for papers for this IJCNN’09 Special Session invited further results on proofs of stability and (rigorous) guarantees of performance for ADP-based designs. In addition, authors were invited to present recent results emphasizing that ADP may in fact join two disparate design methods in feedback control theory; namely, Optimal Control and Adaptive Control (cf. my concluding paragraph in Section II). Further, it was noted that a currently open challenge is to develop appropriate continuous-time procedures and results, because such formulations are known to be more appropriate for important classes of real-world applications. I comment here that once ADP methods that employ NN or Fuzzy components are successfully developed, it appears the path to continuous-time applications is made easier than via the original DP formulation. Finally, the desire was stated in the Call for papers to the community to further develop and rigorously analyze learning and adaptation structures that will allow the on-line design of feedback controllers that are optimal, where both linear and nonlinear systems are to be tractable.

The role of the present paper was to provide a retrospective of ADP to set the stage for the other papers to be presented at the Special Session in response to the above Call. Nevertheless, I wish to add to the above aspirations the Experience Based notions mentioned in Section I of this paper. Success in the “Phase 4” methods described will require solution of all of the above, and in addition, involve a lifting of the perspective of ADP researchers and developers to the “next level up.” The ‘next level up’ phrase refers to the perspective of the practitioner who is employing ADP. Namely, rather than focusing on employing ADP to develop a design for a controller that is optimal (according to the criteria embedded in the Utility Function provided by the intended user), instead, think in terms of already having a collection of previously developed optimal controller designs for a given context (with respective Utility functions), and then employ ADP to develop a strategy for selecting a controller from the collection (‘experience repository’). The resulting ‘selector’ is to be optimal with respect to a Utility function that is to be defined (e.g., relating to the effectiveness and efficiency issues mentioned in Section I, plus the close-loop stability issues previously mentioned) – and we note that formulation of a useful Utility function is to be an item of research itself. This approach of employing ADP to develop an optimal selector (to operate on the experience repository) is in contrast to the usual way of employing ADP to perform an iterative learning procedure to converge upon the optimal design for a controller. Significant issues – both theoretical and conceptual – are there to be solved to achieve the suggested Experience Based approach [31]. I believe this approach parallels what we humans do when we attain and apply experience in a given problem domain.

REFERENCES


