PART I

FEEDBACK CONTROL USING RL AND ADP

CHAPTER 1

Reinforcement Learning and Approximate Dynamic Programming (RLADP)—Foundations, Common Misconceptions, and the Challenges Ahead

PAUL J. WERBOS
National Science Foundation (NSF), Arlington, VA, USA

ABSTRACT

Many new formulations of reinforcement learning and approximate dynamic programming (RLADP) have appeared in recent years, as it has grown in control applications, control theory, operations research, computer science, robotics, and efforts to understand brain intelligence. The chapter reviews the foundations and challenges common to all these areas, in a unified way but with reference to their variations. It highlights cases where experience in one area sheds light on obstacles or common misconceptions in another. Many common beliefs about the limits of RLADP are based on such obstacles and misconceptions, for which solutions already exist. Above all, this chapter pinpoints key opportunities for future research important to the field as a whole and to the larger benefits it offers.

1.1 INTRODUCTION

The field of reinforcement learning and approximate dynamic programming (RLADP) has undergone enormous expansion since about 1988 [1], the year of the first NSF workshop on Neural Networks for Control, which evaluated RLADP as one of several important new tools for intelligent control, with or without neural networks. Since

Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, First Edition. Edited by Frank L. Lewis and Derong Liu.

then, RLADP has grown enormously in many disciplines of engineering, computer science, and cognitive science, especially in neural networks, control engineering, operations research, robotics, machine learning, and efforts to reverse engineer the higher intelligence of the brain. In 1988, when I began funding this area, many people viewed the area as a small and curious niche within a small niche, but by the year 2006, when the Directorate of Engineering at NSF was reorganized, many program directors said "we all do ADP now."

Many new tools, serious applications, and stability theorems have appeared, and are still appearing, in ever great numbers. But at the same time, a wide variety of misconceptions about RLADP have appeared, even within the field itself. The sheer variety of methods and approaches has made it ever more difficult for people to appreciate the underlying unity of the field and of the mathematics, and to take advantage of the best tools and concepts from all parts of the field. At NSF, I have often seen cases where the most advanced and accomplished researchers in the field have become stuck because of fundamental questions or assumptions that were taken care of 30 years before, in a different part of the field. The goal of this chapter is to provide a kind of unified view of the past, present, and future of this field, to address those challenges. I will review many points that, though basic, continue to be obstacles to progress. I will also focus on the larger, long-term research goal of building real-time learning systems which can cope effectively with the degree of system complexity, nonlinearity, random disturbance, computer hardware complexity, and partial observability which even a mouse brain somehow seems to be able to handle [2]. I will also try to clarify issues of notation that have become more and more of a problem as the field grows more diverse. I will try to make this chapter accessible to people across multiple disciplines, but will often make side comments for specialists in different disciplines—as in the next paragraph.

Optimal control, robust control, and adaptive control are often seen as the three main pillars of modern control theory. ADP may be seen as part of optimal control, the part that seeks computationally feasible general methods for the nonlinear stochastic case. It may be seen as a computational tool to find the most accurate possible solutions, subject to computational constraints, to the HJB equation, as required by general nonlinear robust control. It may be formulated as an extension of adaptive control which, because of the implicit "look ahead," achieves stability under much weaker conditions than the well-known forms of direct and indirect adaptive control. The most impressive practical applications so far have involved highly nonlinear challenges, such as missile interception [3] and continuous production of carbon–carbon thermoplastic parts [4].

1.2 WHAT IS RLADP?

1.2.1 Definition of RLADP and the Task it Addresses

The term "RLADP" is a broad and an inclusive term, attempting to unite several overlapping strands of research and technology, such as adaptive critics, adaptive dynamic

programming (ADP), approximate dynamic programming (ADP), and reinforcement learning (RL).

Because the history through 2005 was very complex [3, 4], it is easier to focus first on one of the core tasks that ADP attempts to solve. Suppose that we are given a stochastic system defined by:

$$X(t+1) = F(X(t), u(t), e_1(t)), \tag{1.1}$$

$$Y(t) = H(X(t), e_2(t)), \tag{1.2}$$

and our goal at every time t is to pick $u(t)$ so as to maximize:

$$J = \left\langle \sum_{\tau=t}^{T} U(Y(\tau), u(\tau))/(1+r)^{\tau-t} \right\rangle, \tag{1.3}$$

where r is a discount rate or interest rate, which may be zero or greater than zero, T is a terminal time, which may be finite or may be infinity, $X(t)$ represents the actual state of the system ("the objective real world") at time t, $Y(t)$ represents what we directly observe about the system at time t, $u(t)$ represents the actions or control we get to decide on at each time t, U represents our utility function, following the definitions of Von Neumann and Morgenstern [5], $e_1(t)$ and $e_2(t)$ are vectors or collections of random numbers, and $<>$is notation from physics for expectation value.

This task is called a Partially Observed Markov Decision Problem (POMDP), because any system of $X(t)$ governed by Equation (1.1) is a Markov process.

We are asked to develop methods which are general in that they work for any reasonable nonlinear or linear functions F and H, which may also be functions of unknown weights or parameters W. For a true intelligent system, we want to be able to maximize performance for the case where all our knowledge of F and G comes from experience, from the database $\{Y(\tau), u(\tau), \tau = 1 \text{ to } t\}$, and from an "uninformative" prior probability distribution $\Pr(F, H)$ for what they might be [8].

Modern ADP includes any efforts to use, analyze, or develop general-purpose methods to find good approximate answers to this optimization problem, using learning or approximation methods to cope with complexity. Of course, it also includes efforts aimed at the continuous time version of the problem, and hybrid versions with multiple time scales. It also includes efforts to develop general-purpose methods aimed at major special cases of this problem (such as the deterministic case, where there are no vectors e_1 or e_2), or the fully observed case, where $Y = X$), so long as they are useful steps toward the general case, developing the kinds of methods needed for the general case as well, as discussed in Section 1.2.2.

Reinforcement learning (RL) is much older than ADP. As a result, the term RL means different things to different people. RL includes early work by the psychologist Skinner and his followers, such as Harry Klopf, developing models of how animals learn to change their behavior in response to reward (r) and punishment. Some of the

recent work in RL still follows that tradition, using "r" instead of "U," even when the system is intended to solve an optimization problem. Many computer scientists use the term RL to include systems that try to maximize a function $U(u)$ without considering the impact of present actions on future times. A more modern formulation of RL [1] is essentially the same as ADP, except that we are trying to design a system which observes $U(t)$ at each time t, without knowing the function $U(Y, u)$ which underlies it. This is logically just a special case of ADP, since we can add $U(t)$ itself into the list of observed variables included in Y.

Before 1968, research in RL and research related to dynamic programming were two entirely separate areas. Modern ADP dates back, at the earliest, to the 1968 paper [9] in which I first proposed that we can build reinforcement learning systems through adaptive approximation to the Bellman equation, as will be discussed in Section 1.2.2.

In a recent conference on modernizing the electric power grid [10], I heard a key researcher say "We really need new general methods to solve these complex multistage stochastic optimization problems, but ADP does not work so well. We need to develop better methods for this purpose." Logically this does not make sense, because we have *defined* this field to include any such "better methods." The researcher was actually thinking of one particular set of ADP tools, which do not represent the full capabilities of the field as it exists now, let alone in the future.

Equations (1.1)–(1.3) do not yet give a complete problem specification, but first I need to give some more explanations.

The utility function U is our statement of what *we*—the users, system engineers, or policy makers—want this computer system to do for us. It is a statement of our basic value system, our bottom line, and no computer system can tell us what it should be. There has been a huge amount of literature developed on how to choose U [11–13], which is essential to proper use of such tools. Many system engineers have observed that bad outcomes in large engineering projects result from bad choices of U, or failure to have some kind of U in mind, just as often as they do from failure to maximize U effectively over time. If we pick U just to make the optimization problem easy to solve, we very often will end up with a policy that does a poor job of accomplishing what we really care about.

In many practical applications, people say that they want to minimize something, like cost, instead of maximizing something. Of course, we could just set U equal to minus cost, or reverse the signs of the entire discussion with no real change in the mathematics. Here for simplicity I will stick with the positive formulation.

Many computer scientists have simplified the appearance of Equation (1.3) by defining:

$$\gamma = \frac{1}{1+r}, \tag{1.4}$$

where they call γ a "discount factor." Simplifying algebra often has its uses, but it is extremely important to remember what the real starting point here is [7]. The choice of r is part of our statement as users or policy makers of *what we want our system to*

do. It is a key part of our overall utility function [12]. It is crucial in maintaining the connections between economics and the other domains where ADP is relevant.

In general, it is usually much easier to solve a myopic decision problem, where r is large (or T is finite), than to solve a problem with a commitment to the future, where r is zero. Yet the risks of myopia can be seen at many levels, from the instabilities it can cause in traditional adaptive control to the risks of extinction it poses for the human species as a whole in the face of very complex decision problems. In many situations, the best approach is to start by solving the problem for large r, and then ratcheting r down as close to zero as possible, step by step, by using the policies, weights and parameters of the previous step as initial guesses for the next step. This is one example of the general strategy which Barto has called "shaping," [4] and is now often called "transfer learning." This strategy has led to great results in many practical applications (like some of the earlier work of Jay Farrell), but it can also be used in more automated learning systems. In the limit, when r is zero, the basic theorems of dynamic programming need to be modified; that is why much of my earlier work on ADP [13–15] referred to the seminal work of Ron Howard [16] on that general case, rather than the work of Richard Bellman which it was built on.

Furthermore, in Equations (1.1)–(1.3), I have allowed for the possibility that the system state X and the observables Y may be complex structures, made up of continuous variables, discrete variables, or variables defined over a variable structure graph. The problem specification is also incomplete, insofar as I have said nothing about the possibilities for the functions F and H.

There is an important special case of Equation (1.1), in which: (1) X and Y are simply fixed vectors, $\underline{x}$ and $\underline{y}$, for any given learning system, each made up of a fixed number of continuous and (binary) discrete variables; (2) we implicitly assume that F and H are sampled from some kind of "uninformative prior" distribution, favoring smooth functions and so on, which is natural for such vectors, and does not favor strange higher-order symmetry relations between components of the vector. I call this "vector intelligence [2]," and say more about the crucial concept of uninformative priors in a recent talk for the Erdos Lectures series [8]. One of the two great challenges for basic research in RLADP in coming years is to prove theorems showing that certain families of RLADP design are "optimal" in some sense, in making full use of data from limited experience, in addressing the problem of vector intelligence. Of course, we also need to make such general-purpose tools widely available to the larger community, both for conventional and megacore computer hardware.

As recently as 1990 [4], I hoped that the higher intelligence of simple mammal brains could be matched by such an optimal vector intelligence; however, by 1998 [17], I realized there are fundamental general principles at work in those brains, which provide additional capabilities in handling spatial complexity, complex time structure, and a new level of stochastic creativity. This leads to a roadmap for more advanced ADP systems [2, 8], involving, in order: (1) more powerful systems for approximating complicated nonlinear functions, to better address complexity in X; (2) new extensions of the Bellman equation and methods for approximating *these extensions* efficiently, to address multiple time intervals; and (3) at the highest level, tight new coupling of the stochastic capabilities of the prediction system in the brain

to the ADP circuits proper, supporting a higher level of creativity. Exciting as the new Bellman equations are, the issue of spatial complexity is currently the "other main fundamental challenge" for fundamental RLADP in the coming decades, and will itself require an enormous amount of new effort. Of course, these grand challenges also entail many important research opportunities to get us closer to the larger goals. Equally important are the grand challenges of using ADP in "reverse engineering the brain," and in using these methods to maximum benefit in the three crucial areas of achieving sustainability on earth (e.g., via new energy technologies), achieving economically sustainable human settlement of space (e.g., by solving crucial design and control problems in low-cost access to space), and by better supporting "inner space," realizing the full potential of human intelligence [13].

A major part of the work on RLADP and of dynamic programming deals with the special case where our decision-making system or control system can "see everything" in the plant to be controlled, such that $Y = X$. In that case, Equations (1.1)–(1.3) reduce to:

$$X(t+1) = F(X(t), u(t), e(t)), \tag{1.5}$$

$$J = \left\langle \sum_{\tau=t}^{T} U(X(\tau), u(\tau))/(1+r)^{\tau-t} \right\rangle. \tag{1.6}$$

This is called a Markhov Decision Process (MDP). Some of the theoretical literature on POMDP and MDP assumes that X is just a finite integer, between 1 and N, where N is the number of possible states of the system X. That special case might even be called "lookup table intelligence." It yields important theoretical insights but also some pitfalls, similar to the insights and pitfalls which come in physics when people assume, for simplicity, that the Hamiltonian operator H is a finite matrix [18]. Most of the work on POMDP and MDP in engineering (especially the work on practical applications) now assumes that X is actually a vector $\underline{x}$ in a vector space R^n, where n is the number of *state variables*. That work is well represented in this book, and in its two predecessors [4, 6]. (Unfortunately, some practical engineers refer to n at times as the number of "states.") Systems where X is a combination of a vector $\underline{x}$ and a set of discrete or binary variables are usually called "hybrid systems," or, more precisely, hybrid discrete-continuous systems.

Much of the new work on ADP in operations research (e.g., [19–22]) addresses the case where X is a combination of discrete and integer variables, subject to some combination of equality and inequality constraints. In the special case where $T = 1$, this is called a one-stage decision problem or "stochastic program." The deterministic case of that is called a mixed integer program. As of 2012, decisions about who generates electricity, from day to day or from 5 min interval to 5 min interval, to serve the large-scale electric power market, are made by Independent System Operators (ISO) such as PJM (see www.pjm.org), based on new mixed integer linear programming systems, which have proven that they can handle many thousands of variables quickly enough for practical use in real time; however, because power flows are highly

nonlinear, new nonlinear algorithms for alternating current optimal power flow (such as those of Marija Ilic or James Momoh) have demonstrated great improvements in performance. Unfortunately, the power of all these methods in coping with many thousands of variables has depended on the development of general heuristic tricks, developed by insightful intuitive trial and error, which are mostly proprietary and held very tightly as secrets. The more open literature on stochastic programming, using open software systems like COIN-OR, has some important relations to ADP; there is an emerging community in "stochastic optimization" in OR which tries to bring both together, and to explore new stochastic methods for deterministic problems as well.

The current smart grid policy statement from the White House [23] states: "NSF is supporting research to develop a 'fourth generation intelligent grid' that would use intelligent system-wide optimization to better allow renewable sources and pluggable electric vehicles without compromising reliability or affordability [8]." The paper which it refers to [10] describes substantial opportunities for new applications of ADP, at all level of the electric power system, of great importance as part of the larger effort to make a transition to a sustainable global energy system.

1.2.2 Basic Tools—Bellman Equation, and Value and Policy Functions

Dynamic programming and ADP were originally developed for the MDP case, Equations (1.5) and (1.6).

Before we can build systems that learn to solve MDPs, we first need to define more precisely what we mean by "picking $u(t)$ to maximize J." Looking at Equations (1.1–3), you can see that J depends on future choices of u; therefore, we must make some kind of assumption about future choices of u, to state the problem more precisely. Intuitively, we want to pick $u(t)$ *at all times* to maximize J. We want to pick the value of $u(t)$ at time t, so as to maximize the best we can do in future times to keep on maximizing it.

To translate these intuitive concepts into mathematics, we must rely on the concept of a "policy." A policy π is simply a rule for saying what we will do under all circumstances, now and in the future:

$$u(t) = \pi(X(t)). \tag{1.7}$$

In earlier work [9], we sometimes called this a "strategy." In some RLADP systems, we do rely on explicit policies or "controllers" or "action networks," and in some we do not, but the *mathematical concept* of "policy" underlies all of ADP. In all of modern RLADP, we are trying to converge as closely as possible to the performance of the optimal policy, the policy which maximizes J as defined in Equation (1.6).

Following the notation of Bryson and Ho [24], we may define the function J^*:

$$J^* = \max_{\pi} \left\langle \sum_{\tau=t}^{T} U(X(\tau), \pi(X(\tau)))/(1+r)^{\tau-t} \right\rangle. \tag{1.8}$$

This leads directly to the key equation derived by Richard Bellman (in updated notation):

$$J^*(X(t)) = \max_{u(t)} \left\langle U(X(t), u(t)) + J^*(X(t+1))/(1+r) \right\rangle . \tag{1.9}$$

In dynamic programming (DP) proper, the user must specify a function U, the interest rate r, the function F shown in Equation (1.5), and the set of allowed values which $u(t)$ may be taken from. (When there are constraints on u, the Bellman Equation (1.9) takes care of them automatically; for example, it is still valid in the example where u is taken from the subspace of R^n defined by any number of constraints.) With that information, it is possible to solve for the function J^* which satisfies this equation. The original theorems of DP tell us that J^* exists, and that maximizing $U + J^*/(1+r)$ as shown in the Bellman equation gives us an optimal policy. Note, however, that this depends heavily on the assumption that we can observe the entire state vector (or graph) X; *when people use Equation (1.9) directly on partially observed or nonMarkhovian systems, they often end up with seriously inferior performance*. General-purpose ADP software needs to include tools to address this problem.

In control theory, Equation (1.9) is often called the Hamilton–Jacobi–Bellman (HJB) equation, though this can be misleading. Bellman was the first to solve the stochastic problem in Equations (1.5) and (1.6). Hamilton and Jacobi, in physics, derived an equation similar to Equation (1.9), for the deterministic case, where there is no random noise e and no reference to expectation values and no concept of cardinal utility U. Many of the most important results in nonlinear robust control [25] require that we "solve" (or approximate) the full stochastic Bellman's equation, and not just the deterministic special case.

The function $J^*(X)$ is often called the *value function*, and denoted as $V(X)$.

In theory, exact DP should be able to outperform all other methods for addressing Equations (1.5) and (1.6), including all problems in nonlinear robust control, except for just one difficulty—computational cost. The "curse of dimensionality" for exact DP, and with the simpler forms of RL, is well known. In my Harvard Ph.D. proposal of 1972, and in a journal paper published in 1977 [15], I proposed a general solution to this problem: why not *approximate* the function $J^*(X)$ with an approximation function or model $J^\wedge(X, W)$, with tunable weights W, as in the models we use to make predictions in statistics?

I also provided a general algorithm for training $J^\wedge$, which I called heuristic dynamic programming (HDP), which is essentially the same as what Richard Sutton called TD in his well-known work of 1990 [1].

Because HDP allows for any tunable choice of $J^\wedge$, it is possible to write computer code or pseudocode [4] for HDP which gives the user a wide range of choices, including options such as user-specified models (as in statistics packages), elastic fuzzy logic [26], or universal nonlinear function approximators such as Taylor series or neural networks [27, 28].

In the 1980s, I defined the new term "adaptive critic" to refer to any approximator of J^* or of something like J^*, which contains tunable weights or parameters W

and for which we have a general method to train, adapt or tune those weights. More generally, any RLADP system is an adaptive critic system, if it contains such a system to approximate the value function "or something like the value function."

What else would we want to approximate, other than J^* itself?

When X is actually a vector $\underline{x}$ in R^n and F is differentiable, we usually get better results by approximating:

$$\underline{\lambda}(\underline{x}) = \nabla J^*(\underline{x}). \tag{1.10}$$

The $\underline{\lambda}$ vector is fundamental across many disciplines, and is essential to understanding how decisions and control fit together across different fields. For example, in control theory, the components of $\underline{\lambda}$ are often called the "costate variables." In the deterministic case, they may be found by solving the Pontryagin equation, which is closely related to the original Hamilton–Jacobi equation. In Chapter 13 of [4], I showed how to derive an equation for the stochastic case (a stochastic Pontryagin equation), simply by differentiating the Bellman equation I also specified an algorithm, Dual Heuristic Programming (DHP), for training a critic to approximate $\underline{\lambda}$, and showed that it converges to the right answer at least in the usual multivariate linear/stochastic case.

In economics, the "value" of a commodity x_i is its "marginal utility," which is essentially just λ_i; thus the output of a DHP critic is essentially just a kind of price signal. In applications like electric power, the $\underline{\lambda}$ vector is simply a price vector. It fits Dynamic Stochastic General Equilibrium economics better than the conventional "locational marginal cost" now used in pricing electricity, because it accounts for important effects like the impact of present decisions on future scarcity and congestion. For Freudian psychology, λ_i would represent the emotional value or affect attached to a variable or object, which Freud called "cathexis" or "psychic energy."

Early simulations studies verified that DHP has substantial benefits in performance over HDP [29]. Using a DHP critic, Balakrishnan reduced errors in hit-to-kill missile interception by more than an order of magnitude, compared to all previous methods, in work that has reached many applications. Ferrari and Stengel have also demonstrated its power in applications like reconfigurable flight control.

All of this is what one would expect, based on a simple analysis of learning rates, feedback, and the requirements of local control [4].

Nevertheless, HDP does have the advantage of ensuring that the approximation is globally consistent, and of being able to handle state variables that are not continuous. In order to combine the best advantages of DHP and HDP together, I proposed a different way to approximate J^* in 1987 [30], in which we keep updating the weights W so as to reduce the error measure:

$$E = (J^\wedge(X(t+1))/(1+r) - (U(t) + J^\wedge(X(t), W))^2 + \sum_{i=1}^{n} \alpha_i \left(\frac{\partial J^\wedge}{\partial x_i}(X(t+1))/(1+r) - \left(\frac{\partial U(t)}{\partial x_i} + \frac{\partial J^\wedge}{\partial x_j}(X(t), W) \right) \right)^2, \tag{1.11}$$

where n is the number of continuous variables in the state description X. I called this globalized DHP, or GDHP.

Note that I do not include W in the left-hand sides of these terms, the sides representing $J^*(t+1)$. In 1998 [5], I analyzed the stability and convergence properties of all these methods in the linear-quadratic case, with and without noise. When W is included on "both sides," in variations of the methods that I called "Galerkinized," the weights do not converge to the correct values in the stochastic case, though convergence is guaranteed robustly in the deterministic case. In Section 9 of [5], I described new variations of the methods which should possess both strong robust stability and converge to the right answer in the stochastic case; however, it is not clear whether the additional complexity is worthwhile in practical applications, or whether the brain itself possesses that kind of robust stability. For now, it is often best to train a controller or a policy using the original methods, and then verify convergence and stability for the outcome [3].

In the general case, GDHP requires the use of second-order backpropagation to compute all the derivatives [4, 30]. However, Wunsch et al. [31] have proposed a way to train an additional critic to approximate λ, intended to approximate GDHP without the need for second order derivatives. Liu et al. [32] have recently reported new stability results and simulations for GDHP. In applications like operations research, when we restrict our attention to special forms of value function approximator, GDHP reduces to a very convenient and simple form [22].

Besides approximating $J^*(X)$ or $\underline{\lambda}(X)$, it is also possible to approximate:

$$J'(X(t), u(t)) = Q(X(t), u(t)) = U(X(t), u(t)) + \underset{u(t+1)}{\text{Max}} \left\langle J^*(X(t+1))/(1+r) \right\rangle . \tag{1.12}$$

Note that J' and Q are the same thing. In 1989, Watkins used the term "Q" in his seminal Ph.D. thesis [33], addressing the case where X is an integer (a lookup table), in a process called "Q learning." In the same year [34], independently, I proposed the use of universal approximators to approximate J', in action-dependent HDP. Action-dependent HDP was the method used by White and Sofge [4] in their breakthrough control for the continuous production of thermoplastic carbon–carbon parts, a technology which is now of enormous importance to the aircraft industry, as in the recent breakthrough commercial airplane, the Boeing 787. This is also the approach taken in the recent work by Si, with some variation in how the training is done. Action-dependent versions of DHP [4] and GDHP also exist.

In addition to approximating the function J^* (or $\underline{\lambda}$ or Q), we often need to approximate the optimal policy by using an action function, action network or "actor":

$$u(t) = A(X(t), W, e). \tag{1.13}$$

In other words, if we cannot realistically explore the space of all possible policies π, we can use an approximation function or model A, and explore the space of those policies defined by tuning the weights W in that model. As in standard statistics or advanced neural networks, we can also add and delete terms in A automatically, based on what fits in the data. In most applications today, we do not actually include a random term (e) in the action network, but stochastic exploration of the physical

world is an important part of animal learning, and may become more important in challenging future applications.

These fundamental methods are described in great detail in Handbook of Intelligent Control [4]. Many applications and variations and special cases have appeared since, in [6] and in this book, for example. But there is still a basic choice between approximating J^* (as in HDP), approximating $\underline{\lambda}$ (as in DHP) and approximating J^* while accounting for gradient error (as in GDHP), with or without a dependence on the actions $u(t)$ (as in the action-dependent variations).

A more complete review of the early history through 1998, as well as extensive robust stability results extending to the stochastic case, may be found in [35].

1.2.3 Optimization Over Time Without Value Functions

The value function of dynamic programming, $J^*(t)$, essentially represents the value of a whole complex range of possible future trajectories for later times, which cannot be tabulated explicitly because there are so many of them, because of the uncertainty.

But what about the case where there is no uncertainty (no random disturbance e), or where the uncertainty is so simple that we do not need to account for more than a few possible trajectories?

In those kinds of situations, we do not need to use value functions or ADP. We can try to solve for a fixed schedule of actions, $\{u(1), \ldots, u(T)\}$, by calculating the fixed trajectory they lead to, and calculating J explicitly, and minimizing it by use of classical methods. Bryson and Ho [24] give several methods for doing this. Recent work in receding horizon control and model predictive control (MPC) takes the same approach.

In those situations, we can also use the same kind of direct method to calculate the optimal weights W of an action network like Equation (1.13). This may not give as good performance, in theory, as finding the optimal schedule of action, but the resulting action network may carry over better to future decisions when the time horizon T moves further into the future.

Even when this kind of direct method works, the sheer cost of running forward, say, a hundred time points into the future, and optimizing over choices of trajectory, can be a major limiting factor. So long as F is any differentiable function, one can use backpropagation through time to calculate the gradient of J exactly at low cost, and complementary methods to make better use of those gradients (see Chapter 10 of [4]). This is especially easy when F is a neural network model of the plant; this may be called neural model predictive control (NMPC). Widrow has shown that a neural network action network trained by backpropagation through time [1] can learn amazing performance in the task of backing up a truck, both in simulation and on a physical testbed. This is a highly nonlinear task, and he proved that the system trained on a limited set of states could generalize to perform well across the entire range of possible starting states. Suykens et al. [36] have proven that this method offers far stronger robust stability guarantees than traditional neural adaptive control, which offers guarantees similar to traditional linear adaptive control [37]. NMPC has been extremely successful in many applications in the automobile industry, such as

idle speed control at Ford and 15% improvement of mpg in the Prius hybrid [38]. At Neurodimensions (www.nd.com), Curt Lefebvre developed general software for NMPC, working with Jose Principe, and later reported that his intelligent control system is used in 25% by US coal-fired generators [10].

Can ADP work better in some automotive applications than NMPC? The answer is not clear. Using ADP, Sarangapani has shown mpg improvements more like 5–6% compared to the best conventional controllers for conventional and diesel car engines, and a 98% reduction in NO_x emissions. This is simply a different application.

These direct methods are not part of ADP, since they do not approximate the Bellman equation or any of its relatives. But at the same time, they provide a relatively simple, high quality comparison with ADP. Thus, they should be part of any general software package for useful ADP. Many computer scientists would call such methods "direct policy reinforcement learning," and include them in RLADP. There has been impressive success in dextrous robotics and in understanding human motor control using both approaches; Schaal [39] and Atkeson have mainly used the direct methods, while Todorov [40] has used ADP and hybrids of NMPC and ADP.

One of the most important methods in this class is "differential dynamic programming" (DDP) by Jacobson and Mayne [41]. DDP is not really a form of DP or ADP, but it does address the case where stochastic disturbances e exist. Their method for handling e is very rigorous, and very straightforward; it underlies all my own work on the nonlinear stochastic case. In essence, we simply treat the random variables as additional arguments to F and to other functions in the system. Most methods of handling random noise in reinforcement learning are like what statisticians call "unpaired comparisons" [42]; this method is like "paired comparisons," and far more efficient in using limited data and computational resources. More precisely, paired comparisons tend to reduce error by a factor of sqrt(N), where N is the number of simulated cases.

Just as neural networks and backpropagation through time can improve the performance of conventional MPC, they can also be used in Neural DDP—NDDP? DDP itself already includes a propagation of information backwards through time, based on a global Jacobian, but true backpropagation [43] does better by exploiting the structure of nonlinear dynamical systems.

For relatively simple problems, NMPC and DDP can sometimes outperform adaptive critic methods, but they cannot explain or replicate the kind of complexity we see in intelligent systems like the mammal brain. Unlike ADP, they do not offer a true brain-like real-time learning option. Even in using NMPC in receding horizon control, one can often improve performance by training a critic network to evaluate the final state $X(T)$.

1.3 SOME BASIC CHALLENGES IN IMPLEMENTING ADP

Among the crucial choices in using ADP are

- discrete time versus continuous time,
- how to account for the effect of unseen variables,

- offline controller design versus real-time learning,
- "model-based methods" like HDP and DHP versus "model free methods" like ADHDP and Q learning,
- how to approximate the value function effectively,
- how to pick $u(t)$ at each time t even knowing the value function,
- how to use RLADP to build effective cooperative multiagent systems?

Equations (1.1) through (1.11) all formulate the optimization problem in discrete time—from t to $t + 1$, and so on, up to some final time T. This chapter will not discuss the continuous-time versions, in part because Frank Lewis will address that in his sections. In my own work, I have been motivated most of all by the ultimate goal of understanding and replicating the optimization and prediction capabilities of the mammal brain [2, 13]. The higher levels of the brain like the cerebral cortex and the limbic system are tightly controlled by regular "clock signals" broadcast from the nonspecific thalamus, enforcing basic rhythms of about 8 Hz ("alpha") and 4 Hz ("theta"). However, the brain also includes a faster lower-level motor control system, based on the cerebellum, running more like 200 Hz, acting as a kind of responsive slave to the higher system. Like some of Frank Lewis's designs, it does use a 4 Hz feedback/control signal from higher up, even though its actual operation is much faster. But perhaps some version of discrete-time ADHDP would be almost equivalent to Lewis's continuous time method here. Since I do not know, I will focus instead on the three other challenges here.

1.3.1 Accounting for Unseen Variables

Most engineering applications of ADP do not really fit Equations (1.5) and (1.6). Designs which assume that all the important state variables X are observed directly often perform poorly in the real world. In linear/quadratic optimal control, methods to cope with unobserved variables play a central role in practical systems [24]. In neural network control, variations between different engines and robots and generators play a central role [44]. In the brain itself, reconstruction of reality by the cerebral cortex plays a central role [2]. This is why we need to build general systems that address the general case given in Equations (1.1) through (1.3).

What happens when we apply a simple model-free form of ADP, like ADHDP or Q learning, directly to a system governed by Equations (1.1) and (1.2)? If we pick actions u so as to maximize $Q^{\wedge}(Y, u)$, our critic $Q^{\wedge}$ simply does not have the information we need to make the best decisions. The true Q function is a function of X and u, in effect.

The obvious way to do better is to create some kind of updated estimate of X, which may be called $\underline{x}^{\sim}$ (following [24]) or $X^{\sim}$. For example, the real-world successes of White and Sofge in using ADHDP [4] depended on the fact that they used Extended Kalman Filtering (EKF) to create that kind of estimate. They used ADHDP to train a critic which approximated Q (J') as a function of the $X^{\sim}$ and of u.

Notice that we do not really need to estimate the "true" value of X. Engineers sometimes worry that we could simply measure X in different units, and end up with a different function F, which still fits our observations of Y just as well as the original model. This is called an "identifiability" problem. But we do not really need to know the true "units" in which X should be measured. All we need is the *information* in X. More precisely, if we can develop any updated estimate of $g(X)$, where g is any invertible function, and use that as an input to our Critic, then we should be able to approximate J^* or Q as well as we could if we had an estimate of X itself.

There are four standard ways to develop a state estimate R which can be used as the main input to a critic network or action network, in the general nonlinear case:

- extended Kalman Filter (EKF),
- particle Filter,
- training a time-lagged recurrent network (TLRN) to predict X from Y in simulated data [45, 46],
- extracting the output of the recurrent nodes of a neural network used to model the plant (the system to be controlled) [4, 8].

EKF and particle filters are large subjects, beyond the scope of this chapter—but the work of Feldkamp and Prokhorov at Ford [46] strongly suggests that we do not really need to use them here. In simulation studies of automotive engines, they found that TLRNs and particle filters both performed much better than EKF in state estimation, but that TLRNs had much smaller computational cost. They also fit well on the new embedded control chips used in Ford cars.

But estimating X is still not the best way, in the general case.

In linear/quadratic control, it is well known [24] that we can get optimal control by using "dual control," in which $\underline{x}\tilde{}$ estimated by a Kalman Filter is treated as if it were the true state vector. But in the general nonlinear case, it is not so simple. For example, at the ADP conference in Cocoyoc, Mexico, in 2005, I faced a nonlinear optimization problem, in trying to find two small boys who had run off in the area. In theory, I could estimate the center of gravity of their possible locations, and walk straight there... but I was already close to that center of gravity (where they started from). I had to consider the probabilities of places where they might be, and plan to "buy information" on where they might be. In general, in studies of POMDP, it is well known that the optimal action $u(t)$ at any time depends on the entire "belief state," $\Pr(X)$, rather than just the most likely state.

Fortunately, the problem of optimal state estimation also depends on the entire belief state. Even in linear Kalman filtering [24], it is necessary to update matrices like "P" which represent the uncertainties in state estimation. Thus, the successful results in [46] tell us that the *information* in the belief state is encoded somehow into the recurrent nodes of the TLRN. That is necessary, to get to minimum square error in predicting the state variables, which training to least square error enforces. James Lo [47] has proven more general and formal mathematical results supporting this conclusion.

In summary, at the end of the day, it is good enough to use a TLRN or similar predictor [8] to model the plant, and feed its recurrent nodes into the critic and action networks, as described in [4].

1.3.2 Offline Controller Design Versus Real-Time Learning

Many control engineers start from a detailed model of the function F. Their goal is simply to derive a controller, like the action network of Equation (1.13), which optimizes performance or achieves nonlinear robust control [25]. Often, the most practical approach [10] is to maximize a utility function that combines the two objectives, by adding a term which represents value added by the plant to a term, which represents undesired breakdowns. The action network or controller could be anything from a linear controller, to a soft switching controller of settings for PID controllers, to elastic fuzzy logic [26], to a domain-dependent algorithm in need of calibration, to a neural network. The general algorithms for adapting the parameters W of $A(X, W)$ are the same, across all these choices of A. All the basic methods of ADP can be used in this way. In 1986 [48], I described one way to build a general ADP command within statistical software packages like Troll or SAS to make these kinds of capabilities more widely available.

On the other hand, the mammal brain is based entirely on real-time learning, at some level. At each time t, we observe $Y(t)$ and decide on $u(t)$, and adapt the networks in our brain, and move on to the next time period. Traditional adaptive control [36] takes a similar approach. To fully understand and replicate the intelligence of the mammal brain, we need to develop ADP systems that can operate successfully in this mode.

The tradeoffs between offline learning and real-time learning are not so simple as they appear at first. For example, in 1990, Miller [1] showed how he could train a neural network by traditional adaptive control to push an unstable cart forwards around a figure 8 track. When he doubled the weight on the cart, the network learned to rebalance and recover after only two laps around the track. This seemed very impressive, but we can do much better by training a recurrent network offline [44] to perform this kind of task. Given a database of cart behavior across a wide range of variations in weights, the recurrent network can learn to detect and respond immediately to changes in weight, even though it does not observe the weight of the cart directly. This trick, of "learning offline to be adaptive online," underlies the great successes of Ford in many applications. Presumably the brain itself includes a *combination* of recurrent neurons to provide this kind of quick adaptive response (like the "working memory" studied by Goldmann–Rakic and other neuroscientists) along with real-time learning to handle more novel kinds of changes in the world. It also includes some ability to learn from reliving past memories [8], which could also be used to enhance the prediction systems we use in engineering and social science.

Full, stochastic ADP also makes it possible to develop a controller for a new airplane, for example, which does not require the usual lengthy and expensive period of tweaking the controller when real-time data come in from flight tests. One can build a kind of "metamodel" which allows random variables to change coupling

constants and other parameters which are uncertain, and then train the controller to perform well across the entire range of possibilities [49]. This is different, however, from the type of "metamodeling" which is growing in importance in operations research [22].

Strictly speaking, one might ask: "If brains use TLRNs, how can they adapt them in real time?" For engineering today, backpropagation through time (BTT) is the practical method to use, but for strict real-time operation one may use an "error critic" (Chapter 13 of [4]) to approximate it.

1.3.3 "Model-Based" Versus "Model Free" Designs

DHP requires some kind of model of F, in order to train the λ Critic. Even with HDP, a model of F is needed to find the actions $u(t)$ or train the action network A to maximize $U(t) + \langle J(t+1)\rangle/(1+r)$.

On the other hand, ADHDP and Q learning do not require such a model. This is an important distinction, but its implications are often misunderstood.

Some researchers have even said that the "pure trial and error" character of ADHDP and Q learning make them more plausible as models of how brains work. This ignores the huge literature in animal learning and neuroscience, starting from Pavlov, showing that brains include neural networks which learn to predict or model their environments, and to perform some kind of state estimation as discussed in Section 1.3.1. Because we need state estimation anyway in the general case, it does not cost us much to exploit the information which results from it.

Intuitively, in ADHDP, we choose actions $u(t)$ which are similar to actions which have worked well in the past. In HDP, we pick actions $u(t)$ which are expected to lead to better outcomes at time $t+1$, based on our understanding of what causes what (our model of F). The optimal approach, in principle, is to *combine* both kinds of information. This does require some kind of model of F, but also requires some way to be robust with respect to the uncertainties in that model. For brain-like real-time learning when we cannot use multistreaming [49], this calls for some kind of new hybrid of DHP (or GDHP) and ADHDP.

That will be an important area for research, especially when tools for DHP and HDP proper become more widely available and user-friendly.

Given a straight choice between DHP and ADHDP, the best information we have now [4, 29] suggests that DHP develops more and more advantage as the number of state variables grows. Balakrishnan and Lendaris have done simulation studies showing that the performance of DHP is not so dependent in practice to the details of the model of F. Nevertheless, more research would be useful in providing more systematic and analytical information about these tradeoffs.

In the stochastic case, when we build a model of F by training a neural network or some other universal approximator, we usually train the weights so as to minimize some measure of the error in predicting $Y(t)$ from past data [8]. We assume that the random disturbances added to each variable Y_i are distributed like the errors we see when trying to predict Y_i. When Y_i is a continuous variable, we usually assume that the disturbances follow a Gaussian a distribution. When Y_i is a binary variable, we use

a logistical distribution and error function [42]. This usually works well enough when the sampling time (the difference between t and $t + 1$) is not so large. However, when Y is something complicated, this does not tell us how random disturbances affecting one component of Y correlate with random disturbances in other components. A more general method to model F as a truly stochastic system, accounting for such correlations, is the Stochastic Encoder–Decoder Predictor (SEDP), described in Chapter 13 of [4]. In essence, SEDP is the nonlinear generalization of a method in statistics called maximum likelihood factor analysis. It may also be viewed as a more rigorous and general version of the encoder–decoder "bottleneck" architectures which have shown great success in pattern recognition recently [50–53]; those networks may be seen as the nonlinear generalization of principal component analysis (PCA). I would claim that capabilities like those of SEDP will be necessary in explaining how the mammal brain builds up its model of the world, and that the giant pyramid cells of the cerebral cortex have a unique architecture well-designed to implement that kind of architecture [2].

SEDP and other traditional stochastic modeling tools assume that causality always moves forwards in time. This underlying assumption is implemented by assuming that the random disturbance terms may correlate with later values of the state variables, but not with earlier values [54, 55]. However, a re-examination of the foundations and empirical evidence of quantum mechanics suggests that quantum effects do not fit that assumption [56]. A serious literature now exists on mixed forwards–backwards stochastic differential equations [57]. Human foresight can cause what appear to be backwards causal effects in economic systems, leading to what George Soros calls "reflexive" situations and to multiple solutions in general equilibrium models such as the long-term energy analysis program whose evaluation I once led at the Department of Energy [58]. Hans Georg Zimmermann of Siemens has built neural network modeling systems which allow for time-symmetry in causation, which have been successful in real-world economic and trading applications. However, it would not be easy to build general ADP systems to fully account for time symmetry effects in quantum mechanics, because of the great challenges in building and modeling computing hardware that makes such capabilities available to the systems designer [59]. Hameroff has argued that the intelligence of mammal brains (and earlier brains) may be based on some kind of quantum computing effects, but, like most neuroscientists, I find it hard to believe that such capabilities exist at the systems level in what we can see in mammal brains.

1.3.4 How to Approximate the Value Function Better

Before the time of modern ADP, many people would try to approximate the value function J^* by using simple lookup tables or decision trees. This led to the famous curse of dimensionality. For example, if X consists of 10 continuous variables, and if we consider just 20 possible levels for each of these variables, the lookup table would contain 20^{10} numbers. It would be an enormous, unrealistic computational task to fill in that lookup table, but 20 possible levels might still be too coarse for useful performance.

For a general-purpose ADP system, one would want to be able to approximate J^* or $\underline{\lambda}$ with a universal nonlinear function approximator. Many such universal approximators have been proven to exist, for a large class of possible functions. For example, multivariable Taylor series have often been seen as the most respectable of universal approximators, because of their long history. However, Andrew Barron [27] has proven that all such "linear basis function approximators" show the same basic problem as lookup tables: they require an exponential growth in complexity (number of weights) as the number of state variables grows, for a given quality of approximation of a smooth function. Furthermore, because many polynomial terms correlate heavily with each other, Taylor series approximations usually have severe problems with numerical conditioning [60].

For many applications, then, the best choice for a Critic is the Multilayer Perceptron (MLP), the traditional, simple workhorse of neural network engineering. Barron has proven [27, 28] that the required level of complexity of an MLP grows only as a low power of the number of state variables in approximating a smooth function. This has worked very well in difficult nonlinear applications requiring, say, 10–20 variables. One can use simpler approximations like Gaussians or Radial Basis Functions or differentiable CMAC [4] or kernel methods or adaptive resonance theory in cases where there are fewer state variables or where the system seems to be restricted in practice to a few clusters within the larger state space. All of these neural network approximations also provide an effective way to make full use of the emerging most powerful "megacore" chips. Hewlett–Packard has even developed a software platform which allows use of powerful chips available today, with seamless automatic upgrade to the much more massive capabilities expected in just a few years [61, 62]. There is a reason to suspect that Elastic fuzzy logic [26] might offer performance similar to MLPs in function approximation, since it looks like an exponentiated version of MLP, but this has yet to be proved or disproved.

In 1996, Bertsekas and Tsitsiklis [63] proposed another way to approximate the value function, using user-specified basis functions ϕ_i:

$$J^{\wedge}(X, W) = \sum_{i=1}^{n} W_i \phi_i(X). \tag{1.14}$$

This approximation is still governed, in principle, by Barron's results on linear basis function approximators, but if the users supply basis functions suited to his particular problem, it might allow better performance in practice. This is analogous to those statistical software packages that allow users to estimate models which are "linear in parameters but nonlinear in variables." It is also analogous to the use of user-defined "features," like HOG or SIFT features, in traditional image processing. It also opens the door to many special cases of general-purpose ADP methods, and new special-purpose methods for the linear case, such as the use of linear programming to estimate the weights W [64].

Common sense and Barron's theorems suggest that much better performance could be achieved if the basis functions ϕ_i could be learned, or adaptively improved, instead of being held fixed—but that brings us back, in principle, to the problem of how to

adapt a general nonlinear Critic $J^{\wedge}(X, W)$ or $\underline{\lambda}^{\wedge}(X, W)$, which existing methods already address. In image processing, "deep learning" of neural networks has already led to major breakthroughs, outperforming old feature-based methods developed over decades of intense domain-specific work, and has sometimes reduced error by a factor of two or more compared with traditional methods [50–53]. One would expect deep learning to yield similar benefits here, especially if there is more research in this area.

On the other hand, neurodynamic programming (NDP) as in Equation (1.14) could become an alternative general-purpose method for ADP, if powerful enough methods were found to adapt the basis functions ϕ_i here or in linearized DHP (LDHP), defined by:

$$\lambda_i^{\wedge}(X) = \sum_{j=1}^{n} W_{ij}\phi_j. \tag{1.15}$$

In discussions at an NSF workshop on ADP in Mexico, Van Roy suggested that we could solve this problem by using nonlinear programming somehow. James Momoh suggested that his new implementation of interior point methods for nonlinear programming might make this practical, but there has been no follow-up on this possibility. It leads to technical challenges to be discussed in Section 1.4. Other approaches to this problem have not worked out very well, so far as I know. Of course, it would be easy enough to implement an NDP/HDP hybrid:

- pick the functions ϕ_i themselves to be tunable functions $\phi_i(X, W^{[i]})$,
- adapt the outer weights W_i by NDP,
- treat the outer weights as fixed, adapt the inner weights $\{W^{[i]}\}$ by HDP.

The same could be done with NDP/DHP. However, so far as I know, no one has explored this kind of hybrid.

In presentations at the INFORMS conference years ago, Warren Powell reported that he could get much better results than NDP or deterministic methods, in solving a large-scale stochastic optimization problem from logistics, by using a different simple value function approximator [15]. MLPs are not suitable for tasks like this, in logistics or electric power at the grid level [10], where the number of state variables number in the thousands. They are not a plausible model of Critic networks in the brain, either, because the brain itself can also handle this kind of spatial complexity. In fact, they are not able to handle the level of complexity we see in raw images, in pattern recognition.

To address the challenge of spatial complexity, we now have a whole "ladder" of ever more powerful new neural network designs, starting with "Convolutional Neural Networks" [50–53], going up to Cellular Simultaneous Recurrent Networks (CSRN) [65, 66] and feedforward ObjectNets [67, 68], up to true recurrent object nets [65]. Convolutional Neural Networks are a special case of CSRN and of feedforward ObjectNets, while CSRN are a special case of ObjectNets. The Convolutional Networks,

embodying a few simple design tricks, are the ones that have recently beaten many records in image recognition, phoneme recognition, video, and text analysis. (They should not be confused with the Cellular Neural Networks of Chua and Roska, which are essentially a type of chip architecture.) CSRNs have been shown to do an accurate job of value function approximation for the *generalized* maze navigation problem, a task in which convolutional neural networks failed badly. A machine using a feedforward object net as a Critic was the first system which actually learned master class performance in chess on its own, without a human telling it rules or guidelines for how to play the game, and without a supercomputer [68]. There is every reason to believe that proper use of these new types of neural network could have great power in general-purpose complex applications of ADP. All these network designs are also highly compatible with fast Graphical Processing Unit chips, with Cellular Neural Networks, and with new emerging chips based on memristors [61, 62].

ObjectNets themselves are essentially just a practical approximation of a more complex model of the way in which spatial complexity is handled in the mammal brain and many other vertebrate brains [2, 8].

Grossberg [69] has described models of how the required kind of multiplexing and resetting may be performed, after learning, in the cerebral cortex.

1.3.5 How to Choose $u(t)$ Based on a Value Function

When you have a working estimate of one of the basic value functions, J^* or $\underline{\lambda}$ or Q, the choice of $u(t)$ at time t may be trivial or extremely challenging, depending on the specific application. Of course, as your policy for choosing $u(t)$ changes, your estimate of the value function should change along with it. In my earlier work, I proposed concurrent adaptation of the critic network and the action network (and the network which models the world), as in the brain.

In some applications, even when F represents a serious and challenging engineering plant, the choice of actions $u(t)$ may be just a short list of discrete possibilities. For example, Liu et al. [70] describe a problem in optimal battery management where the task at each time t is to choose between three possible actions:

- charge the battery,
- discharge the battery,
- do neither.

This decision may be very difficult, because it may depend on what we expect the price of electricity to do, and it may depend on things like changing weather or driving plans if the battery is located in a home or a car. It may require a sophisticated stochastic model of price fluctuations and a model of battery lifetime. But despite that complexity, we may still face a simple choice in the end, at each time. In that case, the Q function would be very complex, but we can still just compute $Q^\wedge(X, u)$ for each of the three choices for u, and pick whichever gives the highest value of $Q^\wedge$. If we use a J^* critic, trained by HDP or GDHP, we can still use our model of the system to predict $U(t) + \langle J^*(t+1)/(1+r)\rangle$ for each of the three options, and pick the option

which scores highest. The recent work by Powell and by Liu on the battery problem suggests that the quality of the control algorithm may often be crucial in deciding whether a new battery is money-maker or a money-loser for the people who buy it. Of course, Liu et al. used ADHDP rather than Q learning, because the state vector included continuous variables.

The choice of u is also relatively simple in cases where the sample time is relatively brief, such that the state X is not likely to change dramatically from time t to $t + 1$, for feasible control actions $u(t)$. In such situations, Equation (1.5) may be represented accurately enough by an important special case:

$$\underline{x}(t + 1) = F(\underline{x}(t)) + G(\underline{x}(t))\underline{u}(t), \tag{1.16}$$

with:

$$U(\underline{x}(t), \underline{u}(t)) = U_1(\underline{x}(t)) - c\underline{u}^T(t)R\underline{u}(t). \tag{1.17}$$

This is a variation of the "affine control problem" well-known in control theory. Intuitively, the penalty term u^TRu prevents us from using really large control vectors which would change $\underline{x}$ dramatically over one sample time. In many physical plants, like airplanes or cars, our controls are limited in any case, and it costs some energy to step on the gas too hard. If we use a $\underline{\lambda}$ Critic, as in DHP, the critic already tells us which way we want to move in state space. The optimal policy is simply to move in that direction, as fast as we reasonably can, limited by the penalty term. Using our estimate of $\underline{\lambda}$ from DHP, we can simply solve directly for the optimal value of $\underline{u}(t)$, by algebraically minimizing the quadratic function of $\underline{\lambda}$, G and R implied here. This trick is the basis of Balakrishnan's Single Network Adaptive Critic (SNAC) system [71], which has been studied further by Sarangapani [72]. Many of the recent stability theorems for real-time ADP have also focused on this case of affine control. Note that Balakrishnan's recent results with SNAC, which substantially outperform all other methods in the much-tested application of hit-to-kill missile interception, still use DHP for the training of the Critic.

For the more general case shown in Equation (1.5), we can simply train an action network $A(X, W)$ using the methods given in [4] and in many papers by researchers using that approach. Intuitively, the idea is to train the parameters W so that the resulting $u(t)$ performs the maximization shown in the Bellman Equation (1.9). Regardless of whether A is a neural network or some other differentiable system, backpropagation can be used to calculate the gradient of the function to be maximized with respect to all of the weights, in an efficient and accurate closed form calculation [42]. So long as this maximization problem is convex, a wide variety of gradient-based methods should converge to the correct solution.

When the optimization problem for $u(t)$ in Equation (1.9) is very complicated and nonconvex, more complicated methods may be needed. This often happens when the time interval between t and $t + 1$ is large. For example, in large logistic problems, $u(t)$ may represent a plan of action for day t, and T may be chosen to be $t + 7$, a

week-ahead planning problem. There are several possible approaches here, some available today and some calling for future research.

In electric power, Marija Ilic has recently shown that large savings are possible if large-scale system operators calculate optimal power, voltage and frequency instructions for all generators across the system, every 5 min or so, using a nonlinear one-stage optimization system called "AC optimal power flow" (ACOPF). James Momoh also developed an ACOPF system for the Electric Power Research Institute many years ago [6]. These ACOPF systems already provide good practical approximate solutions to the highly complex, nonlinear problem of maximizing some measure of utility in this domain. One can apply ADP here simply by training a Critic (like an ObjectNet trained by DHP or GDHP), and using the existing ACOPF package to find the optimal $u(t)$. In effect, this adds a kind of foresight capability to the ACOPF decisions. Momoh and I have called this "dynamic stochastic OPF" (DSOPF) [6].

Venayagamoorthy has recently developed a parallel chip implementation of his ObjectNet ADP system [66], which he claims is fast enough to let us unify the new ACOPF capabilities of decisions made every 15 min. with grid regulation decisions which must be made every 2 s, to stabilize frequency and phase and voltage. If this is possible, it would allow local value signals (like the λ outputs of DHP, which value specific outputs form specific units, like price signals in economics) to manage the power electronics of renewable energy systems, so that they shift from being a major cost to the grid to a major benefit; if so, this would have major implications for the economics of renewable energy.

Nonconvexity becomes especially important for biological brains that address the problem of optimization over multiple time intervals [2, 17]. These are similar to robots which need to decide on when to invoke "behaviors" or "motor primitives"; such decisions have a direct impact which goes all the way from time t to the completion of the action, not just to $t + 1$. Use of Action Networks, without additional systems, can lead to suboptimal decisions for such behaviors. In practical terms, this simply implies that our controller has lack of creativity in finding new, out of the box (out of the current basin of attraction) options for higher-level decisions. In [2], I proposed that the modern mouse is more creative than the dinosaur, because of a new system for cognitive mapping of the space of possible decisions, exploiting certain features in its six-layer cerebral cortex that do not exist in the reptile. This is an important area for research, but several steps beyond the useful systems we can build today. It is not even clear how much creativity we would want our robots to have. For the time being, ideas such as stochastic optimization, metamodelling [48], brain-like stochastic search and Powell's work on the exploration gradient present more tangible opportunities.

Nonconvexity can be especially nasty when we are developing multistage versions of problems which are currently treated as single-stage linear problems using Mixed Integer Linear Programming (MILP). Unlike ACOPF, the MILP packages do not have the ability to maximize a general *nonlinear* utility function. Thus for many applications today, the easiest starting point for overcoming myopia and using ADP is to use the existing packages as the Action Network, and train a Critic network to define

the actual objective function which the single-stage optimizer is asked to maximize [19, 22].

It is difficult to document and prove the performance tradeoffs in this case, because the leading MILP system (Gurobi)—like the ACOPF systems of Ilic and Momoh—is highly proprietary. Within linear programming, two major methods have competed through the years—the classical simplex method and the the interior point method pioneered by Karmarkar and Shanno [73]. The simplex method is very specific to the linear case. There are parallelized versions that make effective use of computers with 1–14 processors or so. The interior point method is more compatible with massively parallel processors and with nonlinearity. Shanno even suggested in 1988 [1] that some varieties or adaptations of interior point methods might work better than anything known today to speed up offline learning in neural networks. At that time, interior point was also beginning to perform better on large-scale linear programming problems. But in recent years, Gurobi has developed a variety of highly proprietary heuristics for using simplex for MILP, which outperform what they have for interior point.

Breakthrough performance in this area may well require a new emphasis on interior point methods, using open-source packages like COIN-OR on new more massively parallel computing platforms [62, 63]. Because major users of MILP worry a lot more about physical clock time than they do about the number of processors, massively parallel interior point methods should be able to overtake the classical methods now in use based on simplex. The nonlinear versions of these methods should interface well with nonlinear critics and with neural networks, both in physical interface and at the mathematical level.

1.3.6 How to Build Cooperative Multiagent Systems with RLADP

Multiagent systems (MAS) have grown more popular and more important in recent years. In practice, there are two general types of multiagent systems:

- systems which use distributed control to maximize some kind of global performance of the system as a whole,
- systems truly intended to balance the goals of different humans, with different goals.

For distributed control, it is extremely important to remember that model networks, action networks, and critics can all be *networks*. In other words, global ADP mathematics automatically gives us a recipe for how to manage and tune a global system made up of widely distributed parts.

In some applications, like electric power [10], it has become fashionable to assign a complex decision problem to a large number of independent agents, commonly agents trained by reinforcement learning. It is common to show by simulation or mathematics how these kinds of systems may converge to a Nash "solution." Unfortunately, many researchers do not understand that a Nash equilibrium is not really an optimal outcome

or solution in the general case. It is well-known in game theory that Nash equilibria are commonly far inferior to Pareto optima, which is what we really want in multiplayer systems. In the design of games or of markets, we do often work hard to get to those special cases where a Nash equilibrium would be close to a Pareto optimum. Of course, economics has a lot to tell us about that kind of challenge.

In artificial systems, one can avoid these difficulties by designing a distributed system to be one large ADP system, mathematically, even though the components function independently. With large markets involving people, like power grids, one can use a DHP critic at the systems operator level [10] (or gradients of a GDHP critic) to determine price signals, which can then be sent to independent ADP agents of homeowners that respond to those prices, to organize a market.

In the special case where there, only two actors, in total opposition to each other, one arrives at a Hamilton Jacobi Isaacs equation (HJI). That case is important to higher-order robust control and to many types of military system. Frank Lewis has recently published results generalizing ADP to that case, and begun exploring ADP to seek Pareto optima in multiplayer games of mixed cooperation and conflict.

There has been substantial progress in the electric power sector recently [10] to avoid pathological gaming and Nash equilibrium effects in the power sector. In a global economy which is currently in a Nash equilibrium, far inferior to the sustainable growth which should be possible, it is interesting to consider how optimization approaches might help in getting us closer to a Pareto optimum. More generally, the large complex networks that move electricity, communications, freight, money, water are among the areas where multiagent extensions of ADP have potential to help.

DISCLAIMER

The views herein represent no one's official views but the chapter was written on U.S. government time.

REFERENCES

1. W.T. Miller, R. Sutton, and P. Werbos, editors. *Neural Networks for Control*, MIT Press, Cambridge, MA, 1990.
2. P. Werbos, Intelligence in the brain: a theory of how it works and how to build it. *Neural Networks*, 22(3):200–212, 2009. Related material is posted at www.werbos.com/Mind.htm.
3. S.N. Balakrishnan, J. Ding, and F.L. Lewis. Issues on stability of ADP feedback controllers for dynamical systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(4):913–917, 2008. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp = &arnumber = 4554210&isnumber = 4567535.
4. D. White and D. Sofge, editors. *Handbook of Intelligent Control*, Van Nostrand, 1992. The prefaces and three chapters in the government public domain are posted at www.werbos.com/Mind.htm#mouse.

5. P. Werbos. *Stable Adaptive Control Using New Critic Designs.* xxx.lanl.gov: adap-org/9810001 (October). 1998
6. J. Si, A.G. Barto, W.B. Powell, and D. Wunsch, editors. *Handbook of Learning and Approximate Dynamic Programming* (IEEE Press Series on Computational Intelligence), Wiley-IEEE Press, 2004.
7. J.V. Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*, Princeton University Press, Princeton, NJ, 1953.
8. P. Werbos. *Mathematical foundations of prediction under complexity*, Erdos Lecture series, 2011. Available at http://www.werbos.com/Neural/Erdos_talk_Werbos_final.pdf
9. P. Werbos. The elements of intelligence. Cybernetica (Namur), No. 3, 1968.
10. P.J. Werbos. Computational Intelligence for the smart grid-history, challenges, and opportunities, *Computational Intelligence Magazine, IEEE*, 6(3):14–21, 2011. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp = arnumber = 5952103&isnumber = 5952082.
11. H. Raiffa. *Decision Analysis*, Addison-Wesley, Reading, MA, 1968.
12. P. Werbos. Rational approaches to identifying policy objectives. *Energy: The International Journal*, 15(3/4):171–185, 1990.
13. P. Werbos. Neural networks and the experience and cultivation of mind, *Neural Networks*, 32:86–85, 2012.
14. P. Werbos. Changes in global policy analysis procedures suggested by new methods of optimization, *Policy Analysis and Information Systems*, 3(1): 1979.
15. P. Werbos. Advanced forecasting for global crisis warning and models of intelligence, *General Systems Yearbook*, 1977, p. 37.
16. R. Howard. *Dynamic Programming and Markhov Processes*, MIT Press, Cambridge, MA, 1960.
17. P. Werbos. A brain-like design to learn optimal decision strategies in complex environments. In M. Karny, K. Warwick, and V. Kurkova, editors. *Dealing with Complexity: A Neural Networks Approach*. Springer, London, 1998. Also in S. Amari and N. Kasabov, *Brain-Like Computing and Intelligent Information Systems.* Springer, 1998. See also international patent application #WO 97/46929, filed June 1997, published December 11.
18. T. Kato. *Perturbation Theory for Linear Operators*, Springer, Berlin, 1995.
19. W.B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd edition, Wiley Series in Probability and Statistics, 2011.
20. B. Palmintier, M. Webster, J. Morris, N. Santen, and B. Ustun. 2010. *ADP Toolbox—A Modular System for Rapid Experimentation with Dynamic and Approximate Dynamic Programming*, Massachusetts Institute of Technology, Cambridge, MA.
21. C. Cervellera, A. Wen and V.C.P. Chen. Neural network and regression splinevalue function approximations for stochastic dynamic programming. *Computers and Operations Research*, 34, 70–90, 2007.
22. L. Werbos, R. Kozma, R. Silva-Lugo, G.E. Pazienza, and P. Werbos. Metamodeling and critic-based approach to multi-level optimization, *Neural Networks*, 32:179–185, 2012.
23. A POLICY FRAMEWORK FOR THE 21st CENTURY GRID: *Enabling Our Secure Energy Future* http://www.whitehouse.gov/sites/default/files/microsites/ostp/nstc-smart-grid-june2011.pdf
24. A. Bryson and Y.C. Ho. *Applied Optimal Control*, Ginn, 1969.

25. J.S. Baras and N.S. Patel. Information state for robust control of set-valued discrete time systems, *IEEE, Proceedings of 34th Conference Decision and Control (CDC)*, 1995. p. 2302.
26. P. Werbos. Elastic fuzzy logic: a better fit to neurocontrol and true intelligence. *Journal of Intelligent and Fuzzy Systems*, 1:365–377, 1993. Reprinted and updated in M. Gupta, ed, *Intelligent Control*, IEEE Press, New York, 1995.
27. A.R. Barron. Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Transactions on Information Theory*, IT-39:930–944, 1993.
28. A.R. Barron. Approximation and estimation bounds for artificial neural networks, *Machine Learning*, 14(1):113–143, 1994.
29. D.V. Prokhorov, R.A. Santiago, and D.C. Wunsch II. Adaptive critic designs: a case study for neurocontrol, *Neural Networks*, 8(9):1367–1372, 1995.
30. P. Werbos. Building and understanding adaptive systems: a statistical/numerical approach to factory automation and brain research, *IEEE Transactions of SMC*, 17(1): 1987.
31. D.V. Prokhorov and D.C. II Wunsch. Adaptive critic designs. *Neural Networks, IEEE Transactions on*, 8(5):997–1007, 1997. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp = &arnumber = 623201&isnumber = 13541.
32. D. Liu, D. Wang, and D. Zhao. Adaptive dynamic programming for optimal control of unknown nonlinear discrete-time systems, *2011 IEEE Symposium on, Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, vol., no., pp. 242–249, 2011 11–15 April doi: 10.1109/ADPRL.2011.5967357 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=& arnumber=5967357& isnumber=5967347.
33. C.J.C.H., Watkins. Learning from delayed rewards. Ph.D. thesis, Cambridge University, 1989.
34. P. Werbos. Neural networks for control and system identification. In: IEEE Conference on Decision and Control (Florida), IEEE, New York. 1989.
35. P. Werbos. *Stable Adaptive Control Using New Critic Designs.* xxx.lanl.gov: adap-org/9810001(October 1998).
36. J.A. Suykens, B. DeMoor, and J. Vandewalle. NLQ theory: a neural control framework with global asymptotic stability criteria *Neural Networks*, 10(4):615–637, 1997.
37. K. Narendra and A. Annaswamy. *Stable Adaptive Systems*, Prentice-Hall, Englewood, NJ, 1989.
38. D. Prokhorov. Prius HEV neurocontrol and diagnostics. *Neural Networks*, 21(2–3):458–465, 2008.
39. S. Schaal. Learning Motor Skills in Humans and Humanoids, plenary talk presented at International Joint Conference on Neural Networks 2011 (IJCNN2011). Video forthcoming from the IEEE CIS Multimedia tutorials center, currently http://ewh.ieee.org/cmte/cis/mtsc/ieeecis/video_tutorials.htm.
40. E. Todorov. Efficient computation of optimal actions. *PNAS* 106:11478–11483, 2009.
41. D. Jacobson and D. Mayne. *Differential Dynamic Programming*, American Elsevier, 1970.
42. T.H. Wonnacott and R.J. Wonnacott. *Introductory Statistics for Business and Economics*, 4th edition, Wiley, 1990.

43. P. Werbos. Backwards differentiation in AD and neural nets: Past links and new opportunities. In H.M. Bucker, G. Corliss, P. Hovland, U. Naumann, and Boyana Norris, editors. *Automatic Differentiation: Applications, Theory and Implementations*, Springer, New York, 2005.

44. P. Werbos. Neurocontrollers. In J. Webster, editor. *Encyclopedia of Electrical and Electronics Engineering*, Wiley, 1999.

45. Y.H. Kim and F.L. Lewis. *High-Level Feedback Control with Neural Networks,* World Scientific Series in Robotiocs and Intelligent Systems, Vol. 21, 1998.

46. L.A. Feldkamp and D.V. Prokhorov. Recurrent neural networks for state estimation, in. Proceedings of the Workshop on adaptive and learning systems, Yale University (Narendra ed.), 2003. Posted with authors' permission at http://www.werbos.com/FeldkampProkhorov2003.pdf. Also see http://home.comcast.net/∼dvp/.

47. J.T.-H. Lo. Synthetic approach to optimal filtering. *IEEE Transactions on Neural Networks*, 5(5):803–811, 1994. See also the relaxation of required assumptions in James Ting-Ho Lo and Lei Yu, Recursive Neural Filters and Dynamical Range Transformers, Invited paper, *Proceedings of The IEEE,* 92(3):514–535, March 2004.

48. P. Werbos. Generalized information requirements of intelligent decision-making systems, *SUGI 11 Proceedings*, Cary, NC: SAS Institute, 1986.

49. L. Feldkamp, D. Prokhorov, C. Eagen, and F. Yuan. *Enhanced Multi-Stream Kalman Filter Training for Recurrent Networks*. In J. Suykens and J. Vandewalle, editors. *Nonlinear Modeling: Advanced Black-Box Techniques*. Kluwer Academic, 1998, pp. 29–53. URL: http://home.comcast.net/∼dvp/bpaper.pdf. See also L.A. Feldkamp, G.V. Puskorius, and P.C. Moore, Adaptive behavior from fixed weight networks. *Information Sciences*, 98(1–4):217–235, 1997.

50. K. Kavukcuoglu, P. Sermanet, Y-Lan Boureau, K. Gregor, M. Mathieu, and Y. LeCun. Learning convolutional feature hierachies for visual recognition, *Advances in Neural Information Processing Systems (NIPS 2010)*, 2010.

51. Y. LeCun, K. Kavukvuoglu, and C. Farabet. Convolutional Networks and Applications in Vision, *IEEE Proceedings of International Symposium on Circuits and Systems (ISCAS'10)*, 2010.

52. J. Schmidhuber, Neural network ReNNaissance, plenary talk presented at International Joint Conference on Neural Networks 2011 (IJCNN2011). Video forthcoming from the IEEE CIS Multimedia tutorials center currently at http://ewh.ieee.org/cmte/cis/mtsc/ieeecis/video_tutorials.htm.

53. A. Ng. Deep Learning and Unsupervised Feature Learning, plenary talk presented at International Joint Conference on Neural Networks 2011 (IJCNN2011). Video forthcoming from the IEEE CIS Multimedia tutorials center, currently http://ewh.ieee.org/cmte/cis/mtsc/ieeecis/video_tutorials.htm.

54. G.E.P. Box and G.M. Jenkins. *Time-Series Analysis: Forecasting and Control*, Holden-Day, San Francisco, 1970.

55. D.F. Walls and G.F. Milburn. *Quantum Optics*, Springer, New York, 1994.

56. P. Werbos. Bell's theorem, many worlds and backwards-time physics: not just a matter of interpretation. *International Journal of Theoretical Physics*, 47(11):2862–2874, 2008. URL: http://arxiv.org/abs/0801.1234.

57. N. El-Karoui and L. Mazliak. *Backward Stochastic Differential Equations*. Addison-Wesley Longman, 1997.

58. C.R. Weisbin, R.W. Peelle, and R.G. Alsmiller, Jr. An assessment of The Long-Term Energy Analysis Program used for the EIA 1978 report to Congress, Energy Volume 7, Issue 2, February 1982, pp. 155–170.

59. P. Werbos. Circuit design methods for quantum separator (QS) and systems to use its output. URL: http://arxiv.org/abs/1007.0146.

60. T. Sauer. *Numerical Analysis*, 2nd edition, Addison-Wesley, 2011.

61. G. Snider, R. Amerson, D. Carter, H. Abdalla, M.S. Qureshi, J. Léveillé, M. Versace, H. Ames, S. Patrick, B. Chandler, A. Gorchetchnikov, and E. Mingolla. From synapses to circuitry: using memristive memory to explore the electronic brain, *Computer*, 44(2):21–28, 2011. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp = &arnumber = 5713299&isnumber = 713288.

62. R. Kozma, R. Pino, and G. Pazienza. *Advances in Neuromorphic Memristor Science and Applications*, Springer, 2012.

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

64. D.P. De Farias and B.V. Roy. The linear programming approach to approximate dynamic programming, *Operations Research* 51(6):850–865, 2003. Article Stable URL: http://www.jstor.org/stable/4132447.

65. R. Ilin, R. Kozma, and P.J. Werbos. Beyond backpropagation and feedforward models: a practical training tool for more efficient universal approximator. *IEEE Transactions of Neural Networks* 19(3):929–937, 2008.

66. K.Y. Ren, K.M. Iftekharuddin, and E. White. Large-scale pose invariant face recognition using cellular simultaneous recurrent network, *Applied Optics, Special Issue in Convergence in Optical and Digital: Ettern Recognition*, 49:B92, 2010.

67. S. Mohagheghi, G.K. Venayagamoorthy, and R.G. Harley. Optimal wide area controller and state predictor for a power system. *IEEE Transactions on Power Systems*, 22(2):693–705, 2007.

68. D.B. Fogel, T.J. Hays, S.L. Han, and J. Quon. A self-learning evolutionary chess program. *Proceedings of IEEE*, 92(12):1947–1954, 2004.

69. Y. Cao, S. Grossberg, and J. Markowitz. How does the brain rapidly learn and reorganize view- and positionally-invariant object representations in inferior temporal cortex? *Neural Networks*, 24:1050–1061, 2011.

70. T. H. D. Liu, Residential energy system control and management using adaptive dynamic programming, *IEEE Proceedings of the International Joint Conference on Neural Networks IJCNN2011*, 2011.

71. S. Chen, Y. Yang, S.N. Balakrishnan, N.T. Nguyen, K. Krishnakumar, In: IEEE Proceedings of the International Joint Conference on Neural Networks 2009 (IJCNN2009), 2009.

72. S. Mehraeen and S. Jagannathan. Decentralized near optimal control of a class of interconnected nonlinear discrete-time systems by using online Hamilton–Bellman–Jacobi formulation, *IEEE Transactions on Neural Networks*, 22(11):1709–1722, 2011.

73. J.N.S. Wright. *Numerical Optimization*, Springer, 2006.