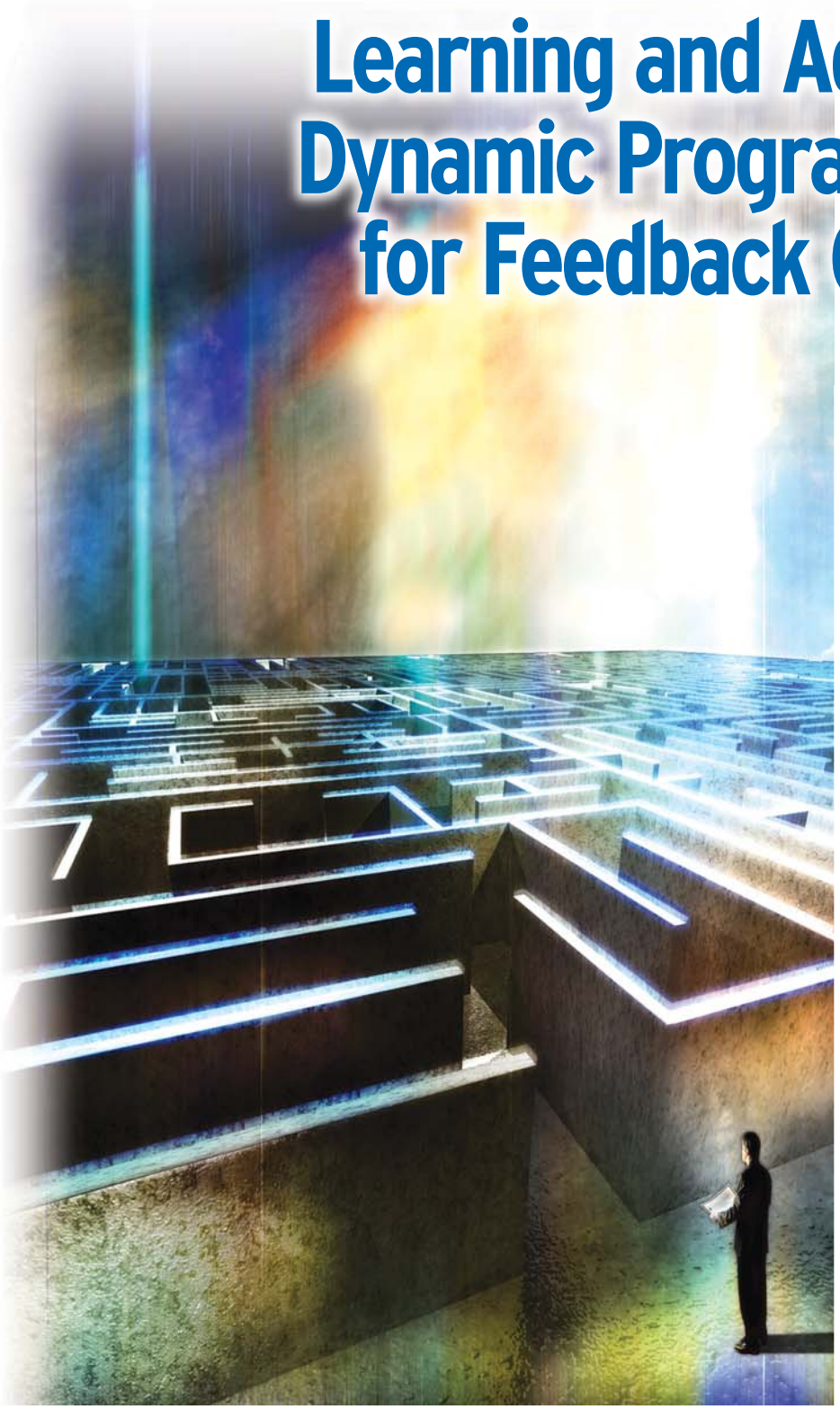
Feature

Reinforcement Learning and Adaptive Dynamic Programming for Feedback Control



Frank L. Lewis
and Draguna Vrabie

Abstract

Living organisms learn by acting on their environment, observing the resulting reward stimulus, and adjusting their actions accordingly to improve the reward. This action-based or Reinforcement Learning can capture notions of optimal behavior occurring in natural systems. We describe mathematical formulations for Reinforcement Learning and a practical implementation method known as Adaptive Dynamic Programming. These give us insight into the design of controllers for man-made engineered systems that both learn and exhibit optimal behavior.

Digital Object Identifier 10.1109/MCAS.2009.933854

 1531-636X/09/$26.00©2009 IEEE

Reinforcement Learning and Optimality in Nature

Every living organism interacts with its environment and uses those interactions to improve its own actions in order to survive and increase. Charles Darwin showed that species modify their actions based on interactions with the environment over long time scales, leading to natural selection and survival of the fittest. Adam Smith showed that modification of the actions of corporate entities based on interactions on the scale of a global economy is responsible for the relative balance and wealth of nations. Ivan Pavlov used simple reinforcement and punishment stimuli to modify the behavior patterns of dogs by inducing conditional reflexes.

We call modification of actions based on interactions with the environment *reinforcement learning (RL)* [Mendel 1970]. There are many types of learning including supervised learning, unsupervised learning, etc. Reinforcement learning refers to an actor or agent that interacts with its environment and modifies its actions, or control policies, based on stimuli received in response to its actions. This is based on evaluative information from the environment and could be called *action-based learning*. RL implies a cause and effect relationship between actions and reward or punishment. It implies goal directed behavior at least insofar as the agent has an understanding of reward versus lack of reward or punishment.

The RL algorithms are constructed on the idea that successful control decisions should be remembered, by means of a reinforcement signal, such that they become more likely to be used a second time. Although the idea originates from experimental animal learning, where it has been observed that the dopamine neurotransmitter acts as a reinforcement informational signal which favors learning at the level of the neuron (see e.g., [Shultz et al. 1997, 2004], [Doya 2001]), RL is strongly connected from a theoretical point of view with direct and indirect adaptive optimal control methods.

One class of reinforcement learning methods is based on the Actor-Critic structure [Barto, Sutton, Anderson 1983], where an actor component applies an action or control policy to the environment, and a critic component assesses the value of that action. Based on this assessment of the value, various schemes may then be used to modify or improve the action in the sense that the new policy yields a value that is improved over the previous value. The actor-critic structure implies two steps: policy evaluation by the critic followed by policy improvement. The policy evaluation step is performed by observing from the environment the results of current actions.

The limits within which organisms can survive are often quite narrow and the resources available to most species are meager. Therefore, most organisms act in an optimal fashion to conserve resources yet achieve their goals. Optimal actions may be based on minimum fuel, minimum energy, minimum risk, maximum reward, and so on. Therefore, it is of interest to study reinforcement learning systems having an actor-critic structure wherein the critic assesses the value of current policies based on some sort of optimality criteria [Werbos 1974, 1989, 1991, 1992], [Bertsekas 1996], [Sutton and Barto 1998], [Cao 2009]. In the optimal RL algorithms case the learning process is moved to a higher level having no longer as object of interest the details of a system's dynamics, but a performance index which quantifies how close to optimality does the closed loop control system operate. In such a scheme, reinforcement learning is a means of *learning optimal behaviors by observing the response from the environment to nonoptimal control policies*.

Feedback Control Theory is the study of means of developing control systems for human engineered systems to endow them with guaranteed performance and safety. Included are control systems for aircraft, ships, race cars, robot systems, industrial processes, building temperature and climate regulation systems, and many more. It is often of interest to mimic nature and design control systems that are optimal in some sense of effectively achieving required performance without using undue amounts of resources.

The purpose of this article is to show the usefulness of reinforcement learning techniques, specifically a family of techniques known as Approximate or Adaptive Dynamic Programming (ADP) (also known as Neurodynamic Programming), for the feedback control of human engineered systems. Reinforcement learning techniques have been developed by the Computational Intelligence Community. Therefore, this requires bringing together ideas from two communities-Control Systems Engineering and Computational Intelligence. Since reinforcement learning involves modifying the control policy based on responses from the environment, one has the initial feeling that it should be closely related to adaptive control, a family of successful control techniques held in high regard in Control Systems Community.

The intention here is to present the main ideas and algorithms of reinforcement learning Approximate Dynamic Programming, not give a literature survey or historical development. Very good surveys are given in [Si et al. 2004], the recent IEEE Transactions on SMC Part B special issue [Lewis, Lendaris, Liu 2008], [Balakrishnan et al.

Frank L. Lewis and Draguna Vrabie are with the Automation & Robotics Research Institute, The University of Texas at Arlington.

2008], and the recent article [Wang, Zhang, Liu 2009]. A biography is included here for further reference by the reader.

Dynamical Systems and Optimal Feedback Control

In the study and design of feedback control systems it is required to provide design algorithms and analysis techniques that yield guaranteed provable performance and safety margins. Feedback controllers without performance, stability, and robustness guarantees will not be accepted by industry. A standard means for providing such guarantees is to use the framework and tools provided by mathematics. Thus, to be precise, we should like to capture the ideas about reinforcement learning in some sort of mathematical formulation. One such formulation is the framework of Markov decision processes (MDP). MDP have been extensively used to study and embody reinforcement learning systems. In MDP, the state spaces and action spaces are generally discrete (i.e. state and action take on only certain allowed discrete values). However, human engineered systems develop and move through time and generally have states and actions that reside in continuous spaces. A broad class of engineered systems can be effectively described by ordinary differential equations, since these describe the development of a system through time based on its current status as well as any inputs received, such as commands, disturbances, and so on.

Dynamical Systems

Physical analysis of dynamical systems using Lagrangian mechanics, Hamiltonian mechanics, etc. produces system descriptions in terms of nonlinear ordinary differential equations. Particularly prevalent are nonlinear ODEs in the state-space form $\dot{x} = f(x, u)$, with the state $x(t) \in R^n$ and control input $u(t) \in R^m$ residing in continuous spaces. Many systems in aerospace, the automotive industry, process industry, robotics, and elsewhere are conveniently put into this form. In addition to being continuous-state space and continuous-input space systems, in contrast to MDP which have discrete states and actions, these dynamics are also continuous-time (CT) systems. For nonlinear systems, the PI algorithm was first developed by Leake and Liu (1967). Three decades later it was introduced in (Beard, Saridis, and Wen, 1997) as a feasible adaptive solution to the CT optimal control problem.

The bulk of research in ADP has been conducted for systems that operate in discrete-time (DT). Therefore, we cover DT systems first, then continuous-time systems. Here, we first consider nonlinear DT systems and outline DT optimal control, developing some computational intelligence notions including policy iteration and value iteration. Then, we illustrate using the linear quadratic regulator (LQR) case to show that these notions are in fact familiar in the feedback control theory setting. After that, we proceed to develop online reinforcement learning schemes for DT dynamical systems. These latter ideas have not been fully exploited in the control systems community.

Optimal Control for Discrete-Time Systems

There are standard methods for sampling or discretizing nonlinear continuous-time state space ODEs to obtain sampled data forms that are convenient for computer-based control [Lewis and Syrmos 1995]. The resulting systems unfold in discrete time and are generally of the state-space form $x_{k+1} = F(x_k, u_k)$ with k the discrete time index. These systems satisfy the 1-step Markov property since their state at time $k + 1$only depends on the state and inputs at the previous time k.

For ease of analysis one often considers a class of discrete-time systems described by nonlinear dynamics in the affine state space difference equation form

$$x_{k+1} = f(x_k) + g(x_k)u_k \tag{1}$$

with state $x_k \in R^n$ and control input $u_k \in R^m$. The analysis of such forms is convenient and can be generalized to the general sampled data form $x_{k+1} = F(x_k, u_k)$.

A *control policy* is defined as a function from state space to control space $h(\cdot):R^n \rightarrow R^m$. That is, for every state x_k, the policy defines a control action

$$u_k = h(x_k). \tag{2}$$

Such mappings are also known as feedback controllers. An example policy is a linear state-variable feedback $u_k = h(x_k) = -Kx_k$. Another example policy is a transfer function dynamical controller design. In the feedback controls community, the feedback control policy can be designed using many methods including optimal control via solution of the Riccati equation, adaptive control, H-infinity control, classical frequency domain control, etc. In reinforcement learning, the control policy is learned in real time based on stimuli received from the environment. Clearly, this learning sort of controller design is related to notions of adaptive control, as we shall see.

System or Environment? In reinforcement learning, the actor is the agent that generates the control policy. That is, the actor is mathematically described by the policy (2), which has the state $x(t)$ as input and the control $u(t)$as output. Everything outside the actor is considered to be the environment. Thus, the system (1) is considered as part of the environment, as are all disturbances and extraneous effects. In fact, in standard

applications of reinforcement learning, the system dynamics is not even considered, and as part of the environment, no explicit model of the dynamics, such as (1), is even used. Reinforcement learning has enjoyed rather remarkable successes for complex systems with unknown dynamics, including the backgammon player of Tesauro, and the design of controllers that can back up truck tractor/trailer rigs with multiple concatenated trailers. However, not specifically considering the dynamics also makes it impossible to provide explicit proofs of stability and performance such as are required for acceptance by the Control Systems Community.

Goal Directed Optimal Performance

The notion of goal-directed optimal behavior is captured by defining a performance measure or cost function

$$V_h(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i) \tag{3}$$

with $0 < \gamma \leq 1$ a discount factor and $u_k = h(x_k)$ a prescribed feedback control policy. This is known as the *cost-to-go* and is a sum of discounted future costs from the current time k into the infinite horizon future. The discount factor reflects the fact that we are less concerned about costs acquired further into the future. Function $r(x_k, u_k)$ is known as the *utility*, and is a measure of the one-step cost of control. This can be selected based on minimum-fuel considerations, minimum energy, minimum risk, etc. For instance, a standard form is the quadratic energy function $r(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k$ or the more general form

$$r(x_k, u_k) = Q(x_k) + u_k^T R u_k, \tag{4}$$

which we use at times for illustration. We require $Q(x)$, R to be positive definite so that the cost function is well defined.

We assume the system is *stabilizable* on some set $\Omega \in R^n$, that is there exists a control policy $u_k = h(x_k)$ such that the closed-loop system $x_{k+1} = f(x_k) + g(x_k)h(x_k)$ is asymptotically stable on Ω. A control policy $u_k = h(x_k)$ is said to be *admissible* if it is stabilizing and yields a finite cost $V_h(x_k)$.

For any admissible policy $u_k = h(x_k)$, we call $V_h(x_k)$ its cost or *value*. Policies with smaller values are deemed to be better than others. It is important to note that, given any admissible policy, its value may be determined by evaluating the infinite sum (3). This may be done by explicit computation in some cases, or by simulation using a digital computer, or by actual evaluation in real-time by observing the trajectories of the closed-loop system.

The means by which the value or cost of a control policy is determined is one of the key differences between feedback control theory and reinforcement learning.

The objective of optimal control theory is to select the policy that minimizes the cost to obtain

$$V^*(x_k) = \min_{h(\cdot)} \left(\sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, h(x_i)) \right) \tag{5}$$

which is known as the optimal cost, or *optimal value*. Then, the optimal control policy is given by

$$h^*(x_k) = \arg\min_{h(\cdot)} \left(\sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, h(x_i)) \right) \tag{6}$$

A short-sighted or myopic planner would only be concerned about minimizing the one-step cost or utility $r(x_k, u_k)$. However, the problem is to minimize not simply the one-step cost, but the sum of all discounted costs, or the cost-to-go. This problem is generally very difficult or even impossible to solve exactly for general nonlinear systems.

Note that in computational intelligence, (3) is often interpreted as a reward, and the objective is to maximize it.

Various methods have been developed to simplify the solution of this optimization problem. Some of these are known within the Control Systems Community and some within the Computational Intelligence Community. We shall discuss:

- Bellman's optimality principle and dynamic programming
- Policy iteration and value iteration
- Various forms of reinforcement learning based on temporal differences and ADP.

Bellman's Optimality Principle and Dynamic Programming

By writing (3) as

$$V_h(x_k) = r(x_k, u_k) + \gamma \sum_{i=k+1}^{\infty} \gamma^{i-(k+1)} r(x_i, u_i), \tag{7}$$

one sees that a difference equation equivalent to is given by

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}),\ V_h(0) = 0. \tag{8}$$

That is, instead of evaluating the infinite sum (3), one can solve the difference equation to obtain the value of using a current policy $u_k = h(x_k)$.

This is a nonlinear Lyapunov equation known as the *Bellman equation*. Evaluating the value of a current policy using the Bellman equation is the first key concept in developing reinforcement learning techniques, as we shall see. Then, we shall show how to solve the Bellman equation on-line in real-time using observed data from the system trajectories.

The DT Hamiltonian can be defined as

$$H(x_k, h(x_K), \Delta V_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}) - V_h(x_k), \tag{9}$$

where $\Delta V_k = \gamma V_h(x_{k+1}) - V_h(x_k)$ is the forward difference operator. The Hamiltonian function captures the energy content along the trajectories of a system as reflected in the desired optimal performance. The Bellman equation requires that the Hamiltonian be equal to zero for the value associated with a prescribed policy.

The optimal value can be written using the Bellman equation as

$$V^*(x_k) = \min_{h(\cdot)}(r(x_k, h(x_k)) + \gamma V_h(x_{k+1})). \qquad (10)$$

This optimization problem is still difficult to solve.

Bellman's principle [Bellman 1957] is a cornerstone of optimal control, and states that "An optimal policy has the property that no matter what the previous decisions (i.e. controls) have been, the remaining decisions must constitute an optimal policy with regard to the state resulting from those previous decisions". In terms of equations, this means that

$$V^*(x_k) = \min_{h(\cdot)}(r(x_k, h(x_k)) + \gamma V^*(x_{k+1})). \qquad (11)$$

This is known as the Bellman *optimality* equation, or the discrete-time Hamilton-Jacobi-Bellman (HJB) equation. One then has the optimal policy as

$$h^*(x_k) = \arg\min_{h(\cdot)}(r(x_k, h(x_k)) + \gamma V^*(x_{k+1})). \qquad (12)$$

Determining optimal controllers using these equations is considerably easier than by using (10), since they contain the optimal value inside the minimization argument.

Since one must know the optimal policy at time $k+1$ to (11) use to determine the optimal policy at time k, Bellman's Principle yields a *backwards-in-time* procedure for solving the optimal control problem. It is the basis for Dynamic Programming algorithms in extensive use in control system theory, Operations Research, and elsewhere. These are by nature *off-line planning methods*. An example of such a procedure in feedback controls design is Riccati equation design for the LQR problem, which involves off-line solution of the Riccati equation given known system dynamics (see below). DP methods generally require the full knowledge of the system dynamical equations. That is $f(x)$, $g(x)$ must be known.

Policy Iteration, Value Iteration, and Fixed Point Equations

In contrast to dynamic programming off-line designs, we seek reinforcement learning schemes for on-line learning in real time, ultimately without knowing the system dynamics $f(x)$, $g(x)$. Therefore, we next show how to exploit the notion that the Bellman equation and the Bellman optimality equation (11) are *fixed point equations* to develop *forward-in-time* methods for solving the optimal control problem.

We are now in a position to use these constructions as a foundation for reinforcement learning optimal control. Consider any given admissible policy $u_k = h(x_k)$ with value $V_h(x_k)$. Motivated, though not justified, by (12) determine a new policy from this value using the operation

$$h'(x_k) = \arg\min_{h(\cdot)}(r(x_k, h(x_k)) + \gamma V_h(x_{k+1})). \qquad (13)$$

This procedure is justified in Bertsekas [1996], where it is shown that the new policy $h'(x_k)$ is *improved* in that it has value $V_{h'}(x_k)$ less than or equal to the old value $V_h(x_k)$. This is known as the one step improvement property of rollout algorithms. That is, the step (13) has given an improved policy.

This suggests the following iterative method for determining the optimal control, which is known as Policy Iteration [Leake and Liu 1967], [Sutton and Barto 1998], [Bertsekas 1996]. See Figure 3.

Policy Iteration (PI) Algorithm

Initialize. Select any admissible (i.e. stabilizing) control policy $h_0(x_k)$

Policy Evaluation Step. Determine the value of the current policy using the Bellman Equation

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1}). \qquad (14)$$

Policy Improvement Step. Determine an improved policy using

$$h_{j+1}(x_k) = \arg\min_{h(\cdot)}(r(x_k, h(x_k)) + \gamma V_{j+1}(x_{k+1})). \qquad (15)$$

If the utility has the special form and the dynamics are (1), then the policy improvement step looks like

$$h_{j+1}(x_k) = -\frac{\gamma}{2} R^{-1} g^T(x_k) \nabla V_{j+1}(x_{k+1}), \qquad (16)$$

where $\nabla V(x) = \partial V(x)/\partial x$ is the gradient of the value function, interpreted here as a column vector.

Note that the initial policy in PI must be admissible, which requires that it be stabilizing. It has been shown by [Leake and Liu 1967], [Howard 1960] and others that this algorithm converges under certain conditions to the optimal value and control policy, that is, to the solution of (11), (12).

The evaluation of the value of the current policy using the Bellman Equation (14) amounts to determining the value of using the policy $h_j(x_k)$ starting in all current states x_k. This is called a full backup in [Sutton and Barto 1998] and can involve significant computation.

In fact, it can be shown that the Bellman equation is a *fixed point equation*. That is, given an admissible policy $u_k = h(x_k)$, has a unique fixed point $V_h(x_k)$, and the following contraction map

$$V^{i+1}(x_k) = r(x_k, h(x_k)) + \gamma V^i(x_{k+1})$$

can be iterated starting with any value $V^0(x_k)$, and there results in the limit $V^i(x_k) \to V_h(x_k)$. Therefore one can replace the policy iteration step (14) by

$$V^{i+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V^i(x_{k+1}), \text{ for } i = 1, 2, \ldots, \quad (17)$$

where the iteration in i is carried out with the same policy $h_j(\cdot)$ until convergence. Then, $V^i(x) \to V_{j+1}(x)$ as $i \to \infty$. One generally selects at step j $V^0(x_{k+1}) = V_j(x_{k+1})$. This can be called Iterative Policy Iteration [Sutton and Barto 1998]. It is noted that each step in (17) is far simpler to implement that a single step of (14), as we see below when we consider the LQR case.

This suggests further the idea of iterating (17) for only K steps, for a fixed finite integer K. That is, only K steps are taken towards evaluating the value of the current policy. This is known as Generalized Policy Iteration in [Sutton and Barto 1998]. In GPI, at each policy update step, only a partial backup is done of the values. An extreme case is to take $K = 1$, which gives the next algorithm, known as Value Iteration. There, only a 1-step backup of values is performed.

Value Iteration (VI) Algorithm

Initialize. Select any control policy $h_0(x_k)$, not necessarily admissible or stabilizing.

Value Update Step. Update the value using

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_j(x_{k+1}). \quad (18)$$

Policy Improvement Step. Determine an improved policy using

$$h_{j+1}(x_k) = \arg\min_{h(\cdot)}(r(x_k, h(x_k)) + \gamma V_{j+1}(x_{k+1})). \quad (19)$$

It is important to note that now, the *old value* is used on the right-hand side of (14), in contrast to the PI step (14). It has been shown that VI converges under certain situations. Note that VI does not require an initial stabilizing policy.

In fact, on further thought, it is seen that Value Iteration is based on the fact that the Bellman Optimality Equation (11) is also a fixed point equation. The interleaved steps of value update and policy improvement are the means of iterating the contraction map associated to (11).

It is important to note that PI requires at each step the solution of (14), which is a nonlinear Lyapunov equation. This solution is difficult for general nonlinear systems. On the other hand, VI relies on the solution of (18), which is simply a recursion equation.

Generally, fixed point equations can be used, with suitable formulation, as a basis for on-line reinforcement learning algorithms that learn by observing data accrued along system trajectories. We shall shortly develop reinforcement learning schemes based on these notions. First, to pin down ideas, let us consider the LQR case.

The DT Linear Quadratic Regulator (LQR) Case

The main purpose of this section is to show that the reinforcement learning notions of Policy Iteration and Value Iteration are in fact in line with familiar ideas in feedback control systems. A second purpose is to give explicit formulae for the above constructions for an important class of problems that illustrates further their meaning.

A large class of important discrete-time (DT) systems can be described in the linear time invariant state-space form

$$x_{k+1} = Ax_k + Bu_k \quad (20)$$

with state $x_k \in R^n$ and control input $u_k \in R^m$. The control policies of interest are then state variable feedbacks of the form

$$u_k = h(x_k) = -Kx_k \quad (21)$$

with the control policy a constant feedback gain matrix K to be determined.

Given a prescribed policy, the cost function is the sum of quadratic functions

$$\begin{aligned} V_h(x_k) &= \sum_{i=k}^{\infty}(x_i^T Q x_i + u_i^T R u_i) \\ &= \sum_{i=k}^{\infty} x_i^T (Q + K^T R K) x_i \equiv V_K(x_k), \end{aligned} \quad (22)$$

which has utility $r(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k$ with weighting matrices $Q = Q^T \geq 0$, $R = R^T > 0$. It is assumed that *(A, B)* is stabilizable, i.e., there exists a feedback gain matrix K such that the closed-loop system

$$x_{k+1} = (A - BK)x_k \equiv A_c x_k \quad (23)$$

is asymptotically stable. It is also assumed that $(A, \sqrt{Q})$ is detectable, i.e. $\sqrt{Q}x_k \to 0$ implies that $x_k \to 0$.

Optimal Control Solution for the DT LQR

The objective of this design is to select the state feedback gain K, i.e. the control policy, to minimize the cost-to-go $V_h(x_k) = V_K(x_k)$ for all current states x_k. This is called the linear quadratic regulator (LQR) problem [Lewis and Syrmos 1995].

It can be shown that the optimal value for the LQR is quadratic in the current state so that

$$V^*(x_k) = x_k^T P x_k \quad (24)$$

for some matrix P, which is to be determined. Therefore, the Bellman equation for the LQR is

$$x_k^T P x_k = x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P x_{k+1}. \quad (25)$$

In terms of the feedback gain this can be written as

$$x_k^T P x_k = x_k^T (Q + K^T R K + (A - BK)^T P (A - BK)) x_k. \quad (26)$$

Since this must hold for all current states x_k one has

$$(A - BK)^T P (A - BK) - P + Q + K^T R K = 0. \quad (27)$$

Every living organism interacts with its environment and uses those interactions to improve its own actions in order to survive and increase.

This matrix equation is linear in P and is known as a Lyapunov equation when K is fixed. Solving this equation given a prescribed stabilizing gain K yields $P = P^T > 0$ such that $V_K(x_k) = x_k^T P x_k$ is the cost of using the policy K. That is,

$$V_K(x_k) = \sum_{i=k}^{\infty} x_i^T (Q + K^T R K) x_i = x_k^T P x_k. \quad (28)$$

The equations and are easily solved for the LQR. Write the Bellman equation as

$$x_k^T P x_k = x_k^T Q x_k + u_k^T R u_k + (A x_k + B u_k)^T P (A x_k + B u_k), \quad (29)$$

whence the minimization is easily performed by differentiating with respect to u_k to obtain

$$R u_k + B^T P (A x_k + B u_k) = 0$$

or

$$u_k = -(R + B^T P B)^{-1} B^T P A x_k, \quad (30)$$

so the optimal feedback gain is

$$K = (R + B^T P B)^{-1} B^T P A. \quad (31)$$

Substituting this into the Bellman equation (29) and simplifying yields the DT HJB equation

$$A^T P A - P + Q - A^T P B (R + B^T P B)^{-1} B^T P A = 0. \quad (32)$$

This equation is quadratic in P and is known as the Riccati equation.

To solve the DT LQR optimal control problem, one first solves the Riccati equation for P, then the optimal value is given by $V^*(x_k) = x_k^T P x_k$ and the optimal policy by (31).

On-line learning vs. off-line planning solution of the LQR. It is important to note the following key point. In going from the formulation (25) of the Bellman equation to the formulation (27), which is the Lyapunov equation, one has performed two steps. First, the system dynamics are substituted for x_{k+1} to yield (26), next the current state x_k is cancelled to obtain (27). These two steps make it impossible to apply real-time online reinforcement learning methods to find the optimal control, which we shall do in the next section. Because of these two steps, optimal controls design in the Control Systems Community is almost universally an off-line procedure involving solutions of Riccati equations, where full knowledge of the system dynamics (A, B) is required.

Policy Iteration and Value Iteration for the DT LQR

We will now see that Policy Iteration and Value Iteration are actually well known in the Control Systems Community, though there they are called something else, namely Hewer's method for solving the DT Riccati equation [Hewer 1971].

For the LQR, Bellman's equation (8) is written as (25) and hence is equivalent to the Lyapunov equation (32). Therefore, in the Policy Iteration Algorithm, the policy evaluation step for LQR is

$$(A - B K_j)^T P_{j+1} (A - B K_j) - P_{j+1} + Q + K_j^T R K_j = 0. \quad (33)$$

and the policy update is

$$K_{j+1} = (R + B^T P_{j+1} B)^{-1} B^T P_{j+1} A. \quad (34)$$

However, iteration of these two equations is exactly Hewer's algorithm [Hewer 1971] to solve the Riccati equation (32). Hewer proved that it converges under the stabilizability and detectability assumptions.

In the Value Iteration Algorithm, the policy evaluation step for LQR is

$$P_{j+1} = (A - B K_j)^T P_j (A - B K_j) + Q + K_j^T R K_j. \quad (35)$$

and the policy update (29) is (34). However, iteration of these two equations has been studied by Lancaster and Rodman [1995], who showed that it converges to the Riccati equation solution under the stated assumptions.

Note that Policy Iteration involves full solution of a Lyapunov equation (33) at each step and requires a stabilizing gain K_j at each step. This is called a full backup in reinforcement learning terms. On the other hand, Value Iteration involves only a Lyapunov recursion (35) at each step, which is very easy to compute, and does not require a stabilizing gain. This is called a partial backup in reinforcement learning.

The recursion (35) can be performed even if K_j is not stabilizing. If K_j is in fact stabilizing, then iterating the Lyapunov recursion (35), with a fixed feedback gain K_j, until convergence provides the solution to the Lyapunov equation (33).

Reinforcement Learning suggests another algorithm for solving the Riccati equation, namely Generalized Policy Iteration. In GPI, one would perform the following at each step.

The means by which the value or cost of a control policy is determined is one of the key differences between feedback control theory and reinforcement learning.

Generalized Policy Algorithm for LQR

Initialize. Select any control policy K_0, not necessarily admissible or stabilizing.

Value Update Step. At step j, update the value using

$$P_j^{i+1} = (A - BK_j)^T P_j^i (A - BK_j) + Q + K_j^T R K_j, \quad i = 0, 1, \ldots, K-1 \tag{36}$$

for some finite K, setting as initial condition $P_j^0 = P_j$. Set $P_{j+1} = P_j^K$.

Policy Improvement Step. Determine an improved policy using

$$K_{j+1} = (R + B^T P_{j+1} B)^{-1} B^T P_{j+1} A. \tag{37}$$

This algorithm takes K steps towards solving the Lyapunov equation at each iteration j. That is, the value update step in GPI consists of K steps of the recursion (35) using the same fixed gain. Setting $K = 1$ yields Value Iteration, i.e. (35), whereas setting $K = \infty$ (i.e. perform (36) until convergence) yields Policy Iteration, which solves the Lyapunov equation (33).

Reinforcement Learning, ADP, and Adaptive Control

The optimal control solution using dynamic programming is a backwards-in-time procedure. Therefore, it can be used for off-line planning but not online learning. We have seen that the Bellman equation (8) leads to several iterative methods for learning the solution of the optimal control equation without solving the HJB equation, including Policy Iteration and Value Iteration. In this section we shall see how to formulate these as *on-line real-time reinforcement learning methods for solving the optimal control problem using data measured along system trajectories* [Sutton and Barto 1998]. These methods are broadly called approximate dynamic programming (ADP) [Werbos 1974, 1989, 1991, 1992] or neurodynamic programming (NDP) [Bertsekas 1996]. There are two key ingredients: temporal difference (TD) error and value function approximation (VFA).

The Bellman equation is a fixed point equation which can be viewed as a consistency equation that the value must satisfy if it is consistent with the current control policy. Generally, fixed point equations can be used, with suitable formulation, as a basis for reinforcement learning algorithms. Let us now develop on-line reinforcement learning schemes based on these notions.

ADP-Temporal Difference (TD) and Value Function Approximation (VFA)

Approximate Dynamic Programming (ADP), or Neurodynamic Programming (NDP), is a practical method for determining the optimal control solution online *forward in time* by using measured system data along the system trajectories. It is based on providing methods for solving the dynamic programming problem forward in time in real-time and for approximating the value function. References are the work of Sutton and Barto [1998], Werbos [1974, 1989, 1991, 1992], and Bertsekas [1996].

Temporal Difference (TD) Error. To turn these concepts into forward-in-time online solution methods, based on the Bellman equation define a time-varying residual equation error as

$$e_k = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}) - V_h(x_k). \tag{38}$$

One notes that the right-hand side of this is the DT Hamiltonian function. Function e_k is known as the temporal difference error. If the Bellman equation holds, the TD error is zero. Therefore, for a fixed control policy $u = h(x)$one may solve the equation $e_k = 0$ at each time k for the value function $V_h(\cdot)$ that is the least-squares solution to the TD equation

$$0 = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}) - V_h(x_k). \tag{39}$$

This yields the best approximation to the value corresponding to using the current policy, i.e. to the summation (3).

The TD error can be considered as a prediction error between predicted performance and observed performance in response to an action applied to the system. See Figure 1.

Figure 1. Reinforcement learning applies an action command and observes the resulting behavior or reward. The difference between the predicted performance and the observed reward plus the current estimate of future behavior is used to modify the action commands to make this difference smaller. This is captured formally in the Bellman Equation.

The usual assumptions made in deriving the Riccati Equation make it impossible to use online learning techniques for the design of optimal feedback control systems.

Solving the TD equation amounts to solving a nonlinear Lyapunov equation on line and without knowing the system dynamics, but using only data measured along the system trajectories. Unfortunately, the TD equation is difficult to solve for general nonlinear systems.

Value Function Approximation (VFA). To provide a practical means for solving the TD equation, one may approximate the value function $V_h(\cdot)$ using a parametric approximator. This has been called Approximate Dynamic Programming (ADP) by Werbos [1974, 1989, 1991, 1992] and neurodynamic programming (NDP) by Bertsekas [1996], both of whom used neural networks as the approximators.

To motivate this approach, let us consider VFA for the LQR case. In LQR, one knows that the value of any admissible control policy $u_k = -Kx_k$ is quadratic in the state, i.e. holds for some matrix P. Substituting into yields the LQR TD error

$$e_k = x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P x_{k+1} - x_k^T P x_k. \tag{40}$$

This equation is linear in the unknown parameter matrix P.

To further simplify the TD equation, use the Kronecker product to write

$$V_K(x_k) = x_k^T P x_k = (vec(P))^T (x_k \otimes x_k) \equiv \overline{p}^T \overline{x}_k \tag{41}$$

with $\otimes$ the Kronecker product and vec(P) the vector formed by stacking the columns of matrix P. Note that $\overline{x}_k = x_k \otimes x_k$ is the quadratic polynomial vector containing all possible products of the n components of x_k. Noting that P is symmetric and has only $n(n+1)/2$ independent elements, one removes the redundant terms in $x_k \otimes x_k$ to define a quadratic basis set $\overline{x}_k$ with $n(n+1)/2$ independent elements. The unknown parameter vector is $\overline{p}$, the elements of matrix P.

Using these constructions, the TD error is written as

$$\begin{aligned} e_k &= x_k^T Q x_k + u_k^T R u_k + \overline{p}^T \overline{x}_{k+1} - \overline{p}^T \overline{x}_k \\ &= r(x_k, u_k) + \overline{p}^T \overline{x}_{k+1} - \overline{p}^T \overline{x}_k. \end{aligned} \tag{42}$$

In the LQR case, a complete basis set for the value function $V_h(x_k)$ is provided by the quadratic functions in the components of x_k. In the nonlinear case one assumes that the value is sufficiently smooth. Then, according to the Weierstrass higher order approximation Theorem, there exists a dense basis set $\{\phi_i(x)\}$ such that

$$\begin{aligned} V_h(x) &= \sum_{i=1}^{\infty} w_i \varphi_i(x) = \sum_{i=1}^{L} w_i \varphi_i(x) + \sum_{i=L+1}^{\infty} w_i \varphi_i(x) \\ &\equiv W^T \phi(x) + \varepsilon_L(x), \end{aligned} \tag{43}$$

where basis vector $\phi(x) = [\varphi_1(x) \quad \varphi_2(x) \quad \cdots \quad \varphi_L(x)]$: $R^n \to R^L$ and $\varepsilon_L(x)$ converges uniformly to zero as the number of terms retained $L \to \infty$. In the Weierstrass Theorem, standard usage takes a polynomial basis set. In the neural network community, approximation results have been shown for various other basis sets including sigmoid, hyperbolic tangent, Gaussian radial basis functions, etc. There, standard results show that the NN approximation error $\varepsilon_L(x)$ is bounded by a constant on a compact set. L is referred to as the number of hidden-layer neurons.

VFA makes several contributions to reinforcement learning. In learning the value function by reinforcement learning methods, it is necessary to store the optimal value and the optimal control as a function of the state vector $x \in R^n$. In Markov Decision Process applications, which are discrete-state systems, e.g., the state can take on only a finite number of prescribed discrete values, this leads to the so-called curse of dimensionality. In CoD, as the number of states and the number of allowed values increases, more and more information must be store, generally in look up tables. However, using value function approximation (VFA), where the critic and, if desired, the actor are parameterized using function approximators, this CoD problem is mitigated.

ADP- On-Line Reinforcement Learning Optimal Control

Assuming the approximation

$$V_h(x) = W^T \phi(x), \tag{44}$$

one substitutes into the Bellman TD equation to obtain

$$e_k = r(x_k, h(x_k)) + \gamma W^T \phi(x_{k+1}) - W^T \phi(x_k). \tag{45}$$

The equation $e_k = 0$ is a fixed point equation. It is a consistency equation that is satisfied at each time k for the value $V_h(\cdot)$ corresponding to the current policy $u = h(x)$. As such, iterative procedures for solving the TD equation may be used, including Policy Iteration and Value Iteration.

On-Line Policy Iteration Algorithm

Initialize. Select any admissible (i.e., stabilizing) control policy $h_0(x_k)$.

Policy Evaluation Step. Determine the least-squares solution W_{j+1} to

$$W_{j+1}^T(\phi(x_k) - \gamma\phi(x_{k+1})) = r(x_k, h_j(x_k)). \quad (46)$$

Policy Improvement Step. Determine an improved policy using

$$h_{j+1}(x_k) = \arg\min_{h(.)}(r(x_k, h(x_k)) + \gamma W_{j+1}^T\varphi(x_{k+1})). \quad (47)$$

If the utility has the special form (4) and the dynamics are (1), then the policy improvement step looks like

$$h_{j+1}(x_k) = -\frac{\gamma}{2}R^{-1}g^T(x_k)\nabla\phi^T(x_{k+1})W_{j+1}, \quad (48)$$

where $\nabla\phi(x) = \partial\phi(x)/\partial x \in R^{L\times n}$ is the Jacobian of the activation function vector.

Note that is a scalar equation, whereas the unknown parameter vector $W_{j+1} \in R^L$ has L elements. At time $k+1$ one measures the previous state x_k, the control $u_k = h_j(x_k)$, the next state x_{k+1}, and computes the utility $r(x_k, h_j(x_k))$. This gives one scalar equation. This is repeated for subsequent times using the same policy $h_j(\cdot)$ until one has at least L equations, at which point one may determine the LS solution W_{j+1}. One may use batch LS for this.

However, note that equations of the form (46) are exactly those solved by recursive least-squares (RLS) techniques. Therefore, one may run RLS online until convergence. Write (46) as

$$W_{j+1}^T\Phi(k) \equiv W_{j+1}^T(\phi(x_k) - \gamma\phi(x_{k+1})) = r(x_k, h_j(x_k)) \quad (49)$$

with $\Phi(k) \equiv (\phi(x_k) - \gamma\phi(x_{k+1}))$ a regression vector. Then, at step j of the PI algorithm, one fixes the control policy at $u = h_j(x)$. Then, at each time k one measures the data set $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$, which consists of the current state, the next state, and the resulting utility incurred. One step of RLS is then performed. This is repeated for subsequent times until convergence to the parameters corresponding to the value $V_{j+1}(x) = W_{j+1}^T\phi(x)$.

Note that for RLS to converge, the regression vector $\Phi(k) \equiv (\phi(x_k) - \gamma\phi(x_{k+1}))$ must be persistently exciting.

As an alternative to RLS, one could use a gradient descent tuning method such as

$$W_{j+1}^{i+1} = W_{j+1}^i - \alpha\,\Phi(k)((W_{j+1}^i)^T\Phi(k) - r(x_k, h_j(x_k)) \quad (50)$$

with $\alpha > 0$ a tuning parameter. The index i is incremented at each increment of the time index k.

Once the value parameters have converged, the control policy is updated according to (47), (48). Then, the procedure is repeated for step $j + 1$. This entire procedure is repeated until convergence to the optimal control solution, i.e., the approximate solution to (11), (12).

This provides an online reinforcement learning algorithm for solving the optimal control problem using Policy Iteration by measuring data along the system trajectories. Likewise, an online reinforcement learning algorithm can be given based on Value Iteration.

On-Line Value Iteration Algorithm

Initialize. Select any control policy $h_0(x_k)$, not necessarily admissible or stabilizing.

Value Update Step. Determine the least-squares solution W_{j+1} to

$$W_{j+1}^T\phi(x_k) = r(x_k, h_j(x_k)) + W_j^T\gamma\phi(x_{k+1}). \quad (51)$$

Policy Improvement Step. Determine an improved policy using

$$h_{j+1}(x_k) = -\frac{\gamma}{2}R^{-1}g^T(x_k)\nabla\phi^T(x_{k+1})W_{j+1}. \quad (52)$$

To solve in real-time one can use batch LS, RLS, or gradient-based methods.

Note that the old weight parameters are on the right-hand side of (51). Thus, the regression vector is now $\phi(x_k)$, which must be persistently exciting for convergence of RLS.

Reinforcement Learning and Adaptive Control

The form of these reinforcement learning algorithms is captured in the figure. Note that they are of the Actor Critic structure. Note that the value update in the critic is performed by solving (46) or (51) using standard adaptive control techniques, namely RLS. Then the control is updated using (52).

Adaptive control can be performed either in a direct fashion, wherein the controller parameters are directly estimated, or in an indirect fashion, wherein the system model parameters are first estimated and then the controller is computed. One sees that reinforcement learning is an indirect adaptive controller wherein the *parameters of the Value* (44) *are estimated*. Then the control is computed using (52). However, the optimal control is directly computed in terms of the learned parameters using (48), so this is actually a *direct adaptive control* scheme!

The importance of reinforcement learning is that it provides an adaptive controller that *converges to the optimal control*. This is new in the Control System Community*, where adaptive controllers do not typically converge to optimal control solutions.* Indirect adaptive controllers have been designed that first estimate system parameters and then solve Riccati equations, but these are clumsy. Reinforcement Learning provides Optimal Adaptive Controllers learned online.

Note that this is a *two-time scale system* wherein the control action in an inner loop occurs at the sampling time, but the performance is evaluated in an outer loop

over a longer horizon, corresponding to the convergence time needed for RLS.

It is important to note that, in the LQR case, the Riccati equation (32) provides the optimal control solution. The Lyapunov equation (27) is equivalent to (45). The dynamics *(A,B)* must be known for the solution of the Lyapunov equation and the Riccati equation. As such these equations provide offline planning solutions. On the other hand, fixed point equation can be evaluated online along the system trajectories using reinforcement learning techniques by measuring at each time the data set $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$, which consists of the current state, the next state, and the resulting utility incurred. This corresponds to *learning the optimal control online by evaluating the performance of nonoptimal controllers.*

Reinforcement learning actually solves the Riccati equation online without knowing the dynamics by observing the data $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$ at each time along the system trajectory.

Introduction of a Second 'Actor' Neural Network

The PI (resp. VI) reinforcement learning algorithm solves a nonlinear Lyapunov equation (resp. Lyapunov recursion) during the Value Update portion of each iteration step *j* by observing only the data set $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$ at each time along the system trajectory. In the LQR PI case, for instance, this means that the Lyapunov (33) equation has been replaced by

$$\overline{p}_{j+1}^T(\overline{x}_{k+1} - \overline{x}_k) = r(x_k, h_j(k)) = x_k^T(Q + K_j^T R K_j)x_k, \quad (53)$$

which is solved for the parameters $\overline{p}_{j+1} = vec(P_{j+1})$ using RLS by measuring the data set $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$ at each time. For this step the dynamics (A, B) can be unknown as they are not needed.

Thus, reinforcement learning solves an underlying nonlinear Lyapunov equation (the Bellman equation) at each step on-line and without knowing the dynamics, by using only data observed along the system trajectories.

However, note that in the LQR case the policy update is given by

$$K_{j+1} = (R + B^T P_{j+1} B)^{-1} B^T P_{j+1} A, \quad (54)$$

which requires full knowledge of the dynamics (A, B). Note further that the embodiment (47) cannot easily be implemented in the nonlinear case because it is implicit in the control, since x_{k+1} depends on $h(\cdot)$ and is the argument of a nonlinear activation function.

These problems are solved by introducing a second neural network for the control policy, known as the actor NN [Werbos 1974, 1989, 1991, 1992]. Therefore, introduce an actor parametric approximator structure

$$u_k = h(x_k) = U^T \sigma(x_k) \quad (55)$$

with $\sigma(x): R^n \to R^M$ a vector of M activation functions and $U \in R^{M \times m}$ a matrix of weights or unknown parameters.

After convergence of the critic NN parameters to W_{j+1} in PI or VI, it is required to perform the policy update (47), (52). To achieve this one may use a gradient descent method for tuning the actor weights U such as

$$U_{j+1}^{i+1} = U_{j+1}^i - \beta \sigma(x_k)(2R(U_{j+1}^i)^T \sigma(x_k) + \gamma g(x_k)^T \nabla \phi^T(x_{k+1}) W_{j+1})^T \quad (56)$$

with $\beta > 0$ a tuning parameter. The tuning index i is incremented with the time index k.

Several items are worthy of note at this point. First, the tuning of the actor NN requires observations at each time k of the data set (x_k, x_{k+1}), i.e., the current state and the next state. However, as per the formulation (55), the actor NN yields the control u_k at time k in terms of the state x_k at time k. Thus, it is a legitimate feedback controller. Second, in the LQR case, the actor NN (55) embodies the gain computation (54). This is highly intriguing, for the latter contains the state internal dynamics A, but the former does not. This means that the A matrix is NOT needed to compute the feedback control. The reason is that the actor NN has learned information about A in its weights, since (x_k, x_{k+1}) are used in its tuning.

Finally, note that only the input function $g(\cdot)$ (in the LQR case, the B matrix) is needed in to tune the actor NN. Thus, introducing a second actor NN has completely avoided the need for knowledge of the state internal dynamics $f(\cdot)$ (or A in the LQR case).

The implementation of reinforcement learning using two NNs, one as a critic and one as an actor, yields the structure shown in Figure 2. In this control system, the critic and the actor are tuned online using the observed data $(x_k, x_{k+1}, r(x_k, h_j(x_k)))$ along the system trajectory. The critic and actor are tuned sequentially in both PI and VI. That is, the weights of one NN are held constant while the weights of the other are tuned until convergence. This procedure is repeated until both NN have converged. Then, the controller has learned the optimal controller online. Thus, this is an *online adaptive optimal control system* wherein the value function parameters are tuned online and the convergence is to the optimal value and control. The convergence of nonlinear Value Iteration using two NN was proven in [Al-Tamimi 2008].

Synchronous methods for tuning the actor and critic are given in [He and Jagannathan 2007]. There, the two NN are tuned simultaneously, and stability of the closed-loop system is guaranteed using Lyapunov energy-based techniques.

Q Learning and Dual Learning

In learning the value function by reinforcement learning methods, the optimal value and the optimal control are stored as a function of the state vector $x \in R^n$. We just saw that, using value function approximation (VFA), where the critic and, if desired, the actor are parameterized using function approximators, this is done in a straightforward manner. The NN weights are tuned online to learn the optimal value and optimal control policy for any value of the state x_k.

Werbos [1974, 1989, 1991, 1992] has introduced four basic methods of approximate dynamic programming (ADP). He has called reinforcement learning based on learning the scalar value function $V_h(x_k)$, Heuristic Dynamic Programming (HDP). Action dependent HDP (AD HDP), introduced as Q learning for discrete-state MDP by Watkins [1989], learns the so-called Q function (also a scalar) and allows one to perform reinforcement learning without any knowledge of the system dynamics. Dual heuristic programming (DHP) uses online learning of the costate function $\lambda_k = \partial V_h(x_k)/\partial x_k$, which is an n-vector gradient and so carries more information than the value. AD DHP is based on learning the gradients of the Q function.

Q Learning

Unfortunately, in value function learning or HDP, one requires knowledge of the system dynamics (see (48) and (54)). At a minimum, one needs the input coupling function $g(\cdot)$ or the B matrix. This is because in performing the minimization (without control constraints)

$$h_{j+1}(x_k) = \arg\min_{h(\cdot)}(r(x_k, h(x_k)) + \gamma W_{j+1}^T \phi(x_{k+1})) \quad (57)$$

one must differentiate with respect to the control to obtain

$$\begin{aligned} 0 &= \frac{\partial}{\partial u_k}(Q(x_k) + u_k^T R u_k) + \frac{\partial}{\partial u_k}\gamma W_{j+1}^T \phi(x_{k+1}) \\ &= 2Ru_k + \left(\frac{\partial}{\partial u_k}\phi(x_{k+1})\right)^T \gamma W_{j+1} \\ &= 2Ru_k + \left(\frac{\partial x_{k+1}}{\partial u_k}\right)^T \nabla\phi^T(x_{k+1})\gamma W_{j+1}. \end{aligned} \quad (58)$$

However in evaluating

$$\frac{\partial x_{k+1}}{\partial u_k} = g(x_k), \quad (59)$$

one requires the system input matrix $g(\cdot)$. If a second actor NN is used, then one still needs $g(\cdot)$ to tune the actor NN weights as per (56).

To avoid knowing any of the system dynamics, one must provide an alternative path to take partial derivatives with respect to the control input that does not go through the system. Werbos has used the concept of backpropagation to accomplish this using action dependent HDP (AD HDP). Watkins [1989] introduced similar notions for discrete-space MDP, which he called Q learning.

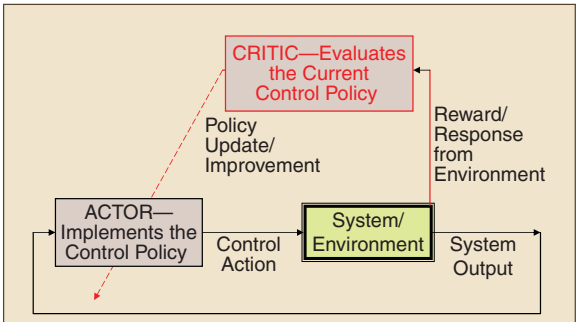


Figure 2. Reinforcement Learning with an actor/critic structure. Using policy evaluation criteria based on optimality mimics nature and also allows for a mathematical formulation that admits rigorous analysis.

Consider the Bellman equation (8), which allows one to compute the value of using any prescribed admissible policy $h(\cdot)$. The optimal control is determined using (10)or (11). Therefore, let us define the Q (quality) function associated with policy $u = h(x)$ as

$$Q_h(x_k, u_k) = r(x_k, u_k) + \gamma V_h(x_{k+1}). \quad (60)$$

Note that the Q function is a function of both the state x_k and the control u_k at time k. It has been called the action value function. Define the optimal Q function as

$$Q^*(x_k, u_k) = r(x_k, u_k) + \gamma V^*(x_{k+1}). \quad (61)$$

In terms of Q^*, one writes the Bellman Optimality equation in the very simple form

$$V^*(x_k) = \min_u(Q^*(x_k, u)) \quad (62)$$

and the optimal control as

$$h^*(x_k) = \arg\min_u(Q^*(x_k, u)). \quad (63)$$

In the absence of control constraints, one obtains the minimum value by solving

$$\frac{\partial}{\partial u}Q^*(x_k, u) = 0. \quad (64)$$

In contrast to (58) this does NOT require any derivatives involving the system dynamics. That is, assuming one knows the Q function for every (x_k, u_k), one does not need to find $\partial x_{k+1}/\partial u_k$.

In value function learning (HDP) one must learn and store the optimal value for all possible states x_k. By contrast, in Q learning, one must store the optimal Q function for all values of (x_k, u_k), that is, for all possible control actions performed in each possible state. This

is far more information. We shall soon see how to use Q function approximation to accomplish this.

Fixed Point Equation for Q Function

We would like to employ online reinforcement techniques to learn the Q function. To do this, we must determine: 1. a fixed point equation for Q to use TD, and 2. a suitable parametric approximator structure for Q to use VFA (actually Q function approximation-QFA).

To determine a fixed point equation for Q, note that

$$Q_h(x_k, h(x_k)) = V_h(x_k). \tag{65}$$

Therefore a 'Bellman equation' for Q is

$$Q_h(x_k, u_k) = r(x_k, u_k) + \gamma Q_h(x_{k+1}, h(x_{k+1})) \tag{66}$$

or

$$Q_h(x_k, h(x_k)) = r(x_k, h(x_k)) + \gamma Q_h(x_{k+1}, h(x_{k+1})). \tag{67}$$

The Optimal Q value satisfies or

$$Q^*(x_k, u_k) = r(x_k, u_k) + \gamma Q^*(x_{k+1}, h^*(x_{k+1})). \tag{68}$$

Equation (67)is a fixed point equation or 'Bellman equation' for Q. Compare it to (8). Now, one can use any online reinforcement learning method from above as the basis for ADP, including PI and VI.

Q Function for LQR Case

To motivate the choice of suitable approximator structures for Q function approximation (QFA), let us compute the Q function for the LQR case.

According to (60) one has for the LQR

$$Q_K(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P x_{k+1}, \tag{69}$$

where P is the solution to the Lyapunov equation for the prescribed policy K. Therefore,

$$Q_K(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k + (Ax_k + Bu_k)^T P (Ax_k + Bu_k) \tag{70}$$

or

$$Q_K(x_k,u_k) = \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} Q + A^T P A & B^T P A \\ A^T P B & R + B^T P B \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \equiv z_k^T H z_k. \tag{71}$$

This is the Q function for LQR. It is quadratic in (x_k, u_k).

Using the Kronecker product one writes

$$Q_K(x_k, u_k) = \overline{H}^T \overline{z}_k$$

with $\overline{H} = \text{vec}(H)$ and $\overline{z}_k = z_k \otimes z_k = \begin{bmatrix} x_k \\ u_k \end{bmatrix} \otimes \begin{bmatrix} x_k \\ u_k \end{bmatrix}$ the quadratic basis set made up from the components of the state and the control input. Then, the fixed point equation (67) is

$$\overline{H}^T \overline{z}_k = x_k^T Q x_k + u_k^T R u_k + \overline{H}^T \overline{z}_{k+1}$$

with $u_k = -Kx_k$. Compare to (42).

Q Function for Reinforcement Learning Using Policy or Value Iteration

Motivated by the LQR example, for nonlinear systems one assumes a parametric approximator or NN of the form

$$Q_h(x, u) = W^T \phi(x, u) = W^T \phi(z) \tag{72}$$

with $\varphi(x, u)$ a basis set of activation functions. This yields the TD error based on (67) of

$$e_k = r(x_k, h(x_k)) + \gamma W^T \phi(z_{k+1}) - W^T \phi(z_k). \tag{73}$$

Now reinforcement learning methods, including PI or VI, may be used to learn $\overline{H} = \text{vec}(H)$ online, exactly as above. Policy Iteration is illustrated below. RLS or gradient-descent can be used to identify the Q function associated to a given policy K as discussed in connection with (46) and (51).

For these methods, the policy update step is based upon

$$\frac{\partial}{\partial u} Q_h(x_k, u) = 0. \tag{74}$$

For the LQR case, define

$$Q_K(x_k,u_k) = z_k^T H z_k = \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} H_{xx} & H_{xu} \\ H_{ux} & H_{uu} \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}. \tag{75}$$

Then, (74) yields

$$0 = H_{ux} x_k + H_{uu} u_k$$

or

$$u_k = -(H_{uu})^{-1} H_{ux} x_k. \tag{76}$$

Since the quadratic kernel matrix H has been found using online reinforcement learning, the system dynamics is not needed for this step. Note that performing (74) on (71) yields exactly (54).

For the general nonlinear case with the critic NN (72) one obtains

$$\frac{\partial}{\partial u} Q_h(x_k, u) = \frac{\partial}{\partial u} W^T \phi(x_k, u) = 0. \tag{77}$$

Since this NN depends explicitly on the control action u (action dependent HDP), the derivatives can be computed without reference to further details such as the system dynamics. To solve for u to obtain an explicit

policy $u_k = h(x_k)$ one requires application of the implicit function theorem to this NN structure.

Both PI and VI can be used for Q Learning. For illustration, we give the

Q Learning Policy Iteration Algorithm

Initialize. Select any admissible (i.e., stabilizing) control policy $h_0(x_k)$.

Policy Evaluation Step. Determine the least-squares solution W_{j+1} to

$$W_{j+1}^T(\phi(z_k) - \gamma\phi(z_{k+1})) = r(x_k, h_j(x_k)). \quad (78)$$

Policy Improvement Step. Determine an improved policy using

$$h_{j+1}(x_k) = \arg\min_{h(.)}(W_{j+1}^T\phi(x_k, u)). \quad (79)$$

The minimization in (79) is given by (77) and can be more explicitly computed given the basis activation functions selected for the NN. For instance, in the LQR case one has (76).

In the policy improvement step one notes a problem. Examine (71) and note that for Q learning one sets $u_k = -Kx_k$ with K the current policy. This makes u_k dependent on x_k and means that the persistence of excitation on $\Phi(k) = (\phi(z_k) - \gamma\phi(z_{k+1}))$ required to solve (78) using LS techniques does not hold. Therefore, one must add a PE probing noise so that $u_k = -Kx_k + n_k$ [Bradtke 1994]. In [Al-Tamimi 2007] it is shown that this does not result in any bias in the Q function estimates.

The resulting structure for reinforcement Q learning is the same as the actor-critic system shown in Figure 2.

It can be shown [Landelius 1997] that for the LQR case, (78) is exactly equivalent to (33), and (79) is the same as (34). Therefore, Q learning effectively solves the Riccati equation online without knowing any system dynamics (A, B).

Several ideas of Werbos [1991, 1992] are intriguing about Q. First, an alternative path has been found to backpropagate the partial derivative $\partial/\partial u_k$ without going through the system dynamics (compare (58) and (77)). Second, the Q function critic NN (72) now has not only the state x_k but also the control action u_k as its inputs. This is the reason $\partial/\partial u_k$ can be evaluated without going through the system. We say the critic NN depends now on the action; Werbos therefore calls this action dependent HDP (AD HDP).

Note that in Q learning, one must store the optimal Q function for all values of (x_k, u_k), in contrast to value learning, where one only stores the optimal value for all values of the state x_k. In MDP this presents an enormous problem of curse of dimensionality. However, VFA (actually QFA) allows one to handle his in a practical manner.

Dual or Gradient Learning

HDP reinforcement Learning methods based on the value can be determined using the Bellman or fixed point equation

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}). \quad (80)$$

AD HDP reinforcement Learning methods based on the Q function can be determined using the Bellman or fixed point equation

$$Q_h(x_k, h(x_k)) = r(x_k, h(x_k)) + \gamma Q_h(x_{k+1}, h(x_{k+1})). \quad (81)$$

Both the value and the Q function are scalars so that learning is being evaluated on the basis of a rather meager response stimulus from the environment and convergence can be slow for systems with large number n of states.

Werbos [1989, 1991, 1992] has proposed using reinforcement learning techniques on the costate

$$\lambda_k = \partial V_h(x_k)/\partial x_k, \quad (82)$$

which is an n-vector gradient and so carries more information than the value. This he called dual heuristic programming (DHP). To perform this, it is necessary to find a fixed point equation for the costate. This is easily done by differentiating to obtain

$$\frac{\partial}{\partial x_k}V_h(x_k) = \frac{\partial}{\partial x_k}r(x_k, h(x_k)) + \frac{\partial}{\partial x_k}\gamma V_h(x_{k+1})$$

or

$$\lambda_k = \frac{\partial r(x_k, u_k)}{\partial x_k} + \left[\frac{\partial u_k}{\partial x_k}\right]^T \frac{\partial r(x_k, u_k)}{\partial u_k} + \gamma\left[\frac{\partial x_{k+1}}{\partial x_k} + \frac{\partial x_{k+1}}{\partial u_k}\frac{\partial u_k}{\partial x_k}\right]^T \lambda_{k+1}$$

for a prescribed policy $u_k = h(x_k)$. Now a NN structure can be used to approximate λ_k and reinforcement learning can proceed.

Unfortunately, any reinforcement learning scheme based on this fixed point equation requires knowledge of the full plant dynamics since $\partial x_{k+1}/\partial x_k = f(x_k)$, $\partial x_{k+1}/\partial u_k = g(x_k)$. Moreover, this requires online RLS implementation for an n-vector, which is computationally intensive.

Similarly, one can find reinforcement learning techniques based on the gradients of the Q function $\lambda_k^x = \partial Q_h(x_k, u_k)/\partial x_k$, $\lambda_k^u = \partial Q_h(x_k, u_k)/\partial u_k$. This is known as AD DHP, and has the same deficiencies just noted for DHP.

Reinforcement Learning and ADP for Continuous-Time Systems

Reinforcement Learning is considerably more difficult for continuous-time systems than for discrete-time systems, and its development has lagged. We shall now see why.

Consider the continuous-time nonlinear dynamical system

$$\dot{x} = f(x) + g(x)u \tag{83}$$

with state $x(t) \in R^n$, control input $u(t) \in R^m$, and the usual mild assumptions required for existence of unique solutions and an equilibrium point to $x = 0$, e.g $f(0) = 0$ and $f(x) + g(x)u$ Lipschitz on a set $\Omega \subseteq R^n$ that contains the origin. We assume the system is stabilizable on Ω, that is there exists a continuous control function $u(t)$ such that the closed-loop system is asymptotically stable on Ω.

The notion of goal-directed optimal behavior is captured by defining a performance measure or cost function associated with the feedback control policy $u = \mu(x)$ as

$$V^{\mu}(x(t)) = \int_t^{\infty} r(x(\tau), u(\tau))d\tau \tag{84}$$

with utility $r(x, u) = Q(x) + u^T Ru$, with $Q(x)$ positive definite, i.e., $\forall x \neq 0$, $Q(x) > 0$ and $x = 0 \Rightarrow Q(x) = 0$, and $R \in \boldsymbol{R}^{m\times m}$ a positive definite matrix.

A policy is called admissible if it is continuous, stabilizes the system, and has a finite associated cost. If the cost is smooth, then an infinitesimal equivalent to (84) can be found by differentiation to be the nonlinear Lyapunov equation

$$0 = r(x, \mu(x)) + (\nabla V^{\mu})^T(f(x) + g(x)\mu(x)), \;\; V^{\mu}(0) = 0, \tag{85}$$

where ∇V^{μ} (a column vector) denotes the gradient of the cost function V^{μ} with respect to x.

This is the CT Bellman equation. It is defined based on the CT Hamiltonian function

$$H(x, \mu(x), \nabla V^{\mu}) = r(x, \mu(x)) + (\nabla V^{\mu})^T(f(x) + g(x)\mu(x)). \tag{86}$$

We now see the problem with CT systems immediately.

Compare the CT Bellman Hamiltonian (86) to the DT Hamiltonian (9). The former contains the full system dynamics $f(x) + g(x)u$, while the DT Hamiltonian does not. This means that there is no hope of using the CT Bellman equation (85) as a basis for reinforcement learning unless the full dynamics are known.

Several studies have been made about reinforcement learning and ADP for CT systems, including [Baird 1994, Doya 2000, Hanselmann 2007, Murray 2001, Mehta and Meyn 2009]. Baird uses Euler's method to discretize the CT Bellman equation. Noting that

$$0 = r(x,\mu(x)) + (\nabla V^{\mu})^T(f(x) + g(x)\mu(x)) = r(x,\mu(x)) + \dot{V}^{\mu} \tag{87}$$

one uses Euler's method to discretize this to obtain

$$\begin{aligned} 0 &= r(x_k, u_k) + \frac{V^{\mu}(x_{k+1}) - V^{\mu}(x_k)}{T} \\ &\equiv \frac{r_S(x_k, u_k)}{T} + \frac{V^{\mu}(x_{k+1}) - V^{\mu}(x_k)}{T} \end{aligned} \tag{88}$$

with sample period T so that $t = kT$. The discrete sampled utility is $r_S(x_k, u_k) = r(x_k, u_k)T$, where it is important to multiply the CT utility by the sample period.

Now note that the discretized CT Bellman equation (88) has the same form as the DT Bellman equation (8). Therefore, all the reinforcement learning methods just described can be applied. Baird defined Advantage Learning based on this as a method of improving the conditioning of reinforcement learning for sampled CT systems. He noted that if the utility is not properly discretized, then the DT solutions do not converge to the CT solutions as T becomes small.

However, this is an approximation only. An alternative exact method for CT reinforcement learning was given in [Vrabie 2009]. One may write the cost in the interval reinforcement form

$$V^{\mu}(x(t)) = \int_t^{t+T} r(x(\tau), u(\tau))d\tau \;\; + \; V^{\mu}(x(t+T)). \tag{89}$$

For any $T > 0$. This is exactly in the form of the DT Bellman equation (8). According to Bellman's principle, the optimal value is given in terms of this construction as [Lewis and Syrmos 1995]

$$V^*(x(t)) = \min_{\overline{u}(t:t+T)} \left(\int_t^{t+T} r(x(\tau), u(\tau))d\tau \;\; + \; V^*(x(t+T)) \right),$$

where $\overline{u}(t{:}t+T) = \{u(\tau){:}t \leq \tau < t+T\}$. The optimal control is

$$\mu^*(x(t)) = \arg\min_{\overline{u}(t:t+T)} \left(\int_t^{t+T} r(x(\tau), u(\tau))d\tau \;\; + \; V^*(x(t+T)) \right).$$

It is shown in [Vrabie 2009] that the nonlinear Lyapunov equation (85) is exactly equivalent to the interval reinforcement form (89). That is, the positive definite solution of both is the value (84) of the policy $u = \mu(x)$.

The interval reinforcement form is a Bellman equation for CT systems, and serves as a fixed point equation. Therefore, one can define the temporal difference error for CT systems as

$$\begin{aligned} e(t{:}t+T) = &\int_t^{t+T} r(x(\tau), u(\tau))d\tau \\ &+ \; V^{\mu}(x(t+T)) \; - \; V^{\mu}(x(t)). \end{aligned} \tag{90}$$

This does not involve the system dynamics.

Now, it is direct to formulate policy iteration and value iteration for CT systems.

CT Policy Iteration (PI) Algorithm

Initialize. Select any admissible (i.e., stabilizing) control policy $\mu^{(0)}(x)$.

Policy Evaluation Step. Solve for $V^{\mu^{(i)}}(x(t))$ using

$$V^{\mu^{(i)}}(x(t)) = \int_t^{t+T} r(x(s), \mu^{(i)}(x(s)))ds + V^{\mu^{(i)}}(x(t+T)) \text{ with } V^{\mu^{(i)}}(0) = 0. \quad (91)$$

Policy Improvement Step. Determine an improved policy using

$$\mu^{(i+1)} = \arg\min_u [H(x, u, \nabla V_x^{\mu^{(i)}})], \quad (92)$$

which explicitly is

$$\mu^{(i+1)}(x) = -\frac{1}{2} R^{-1} g^T(x) \nabla V_x^{\mu^{(i)}}. \quad (93)$$

CT Value Iteration (VI) Algorithm

Initialize. Select any control policy $\mu^{(0)}(x)$, not necessarily stabilizing.

Policy Evaluation Step. solve for $V^{\mu^{(i)}}(x(t))$ using

$$V^{\mu^{(i)}}(x(t)) = \int_t^{t+T} r(x(s), \mu^{(i)}(x(s)))ds + V^{\mu^{(i-1)}}(x(t+T)) \text{ with } V^{\mu^{(i)}}(0) = 0. \quad (94)$$

Policy Improvement Step. Determine an improved policy using

$$\mu^{(i+1)} = \arg\min_u [H(x, u, \nabla V_x^{\mu^{(i)}})], \quad (95)$$

which explicitly is

$$\mu^{(i+1)}(x) = -\frac{1}{2} R^{-1} g^T(x) \nabla V_x^{\mu^{(i)}}. \quad (96)$$

Note that neither algorithm requires knowledge about the internal system dynamics function $f(\cdot)$. That is, they work for partially unknown systems.

Both of these algorithms may be implemented online using the above reinforcement learning techniques. The time is incremented at each iteration by the period T. The measured data at each time increment is $(x(t), x(t+T), \rho(t{:}t+T))$ where

$$\rho(t{:}t+T) = \int_t^{t+T} r(x(\tau), u(\tau))d\tau$$

is the reinforcement measured on each time interval.

The reinforcement learning time interval T need not be the same at each iteration. T can be changed depending on how long it takes to get meaningful information from the observations.

In the LQR case, Policy Iteration is exactly the same as Kleinman's algorithm [Kleinman 1968] for solving the CT Riccati equation. However, these RL methods allow one to implement the algorithm using only information about $g(\cdot)$ (e.g. the B matrix). Information about $f(\cdot)$ (A matrix) is not needed. That is, CT PI solves the CT Riccati equation online without knowing the system internal dynamics by using data measured along the system trajectories.

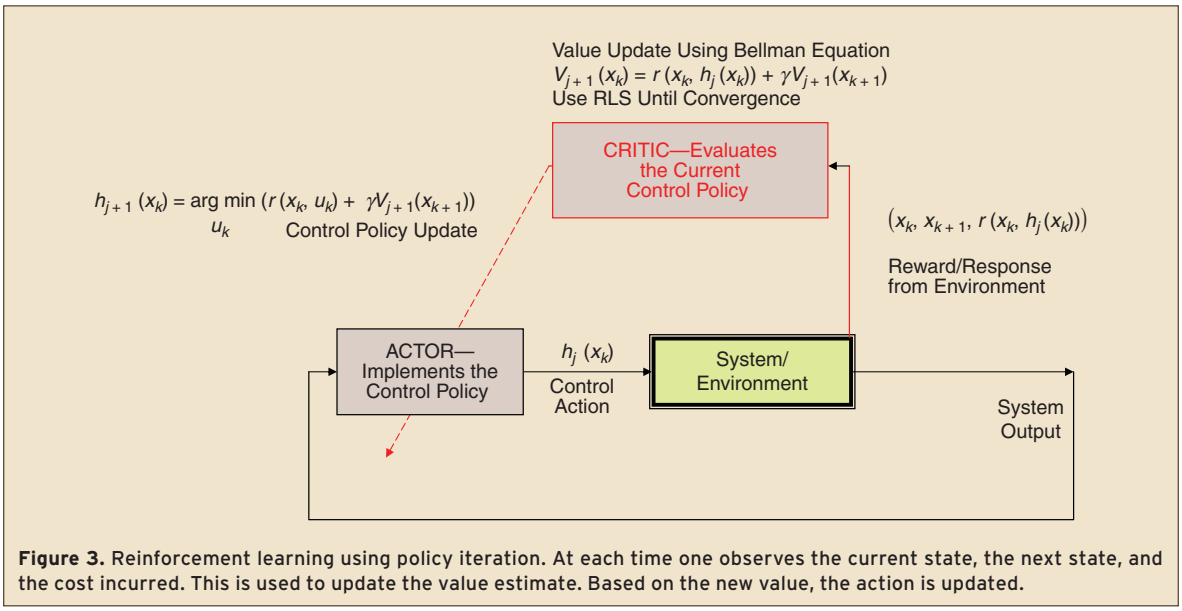


Figure 3. Reinforcement learning using policy iteration. At each time one observes the current state, the next state, and the cost incurred. This is used to update the value estimate. Based on the new value, the action is updated.

Q learning allows one to solve the Riccati equation online in real time without knowing the system description, simply by observing data measured along the system trajectories.

Reinforcement Learning Mechanisms in the Brain

It is interesting to note the similarities between these ADP Reinforcement learning controller structures and learning mechanisms in the mammal brain [Vrabie 2009]. The critic structure learns, in an episodic manner and based on samples of the reward signal from the environment, the parameters of a function which describes the actor performance. Once a performance evaluation episode is completed, the critic passes this information to the actor structure which will use it to adapt for improved performance. At all times the actor must perform continuous-time control for the system (the environment in which optimal behavior is sought). This description of the way in which the actor/critic structure works while searching for optimal control policies points out the existence of two time scales for the mechanisms involved:

- a fast time scale which characterizes the feedback control process at which the actor operates, and
- a slower time scale which characterizes the learning processes at the level of the critic.

Thus the actor and critic structures perform tasks at different operation frequencies in relation with the nature of the task to be performed (i.e., learning or control).

Evidence regarding the oscillatory behavior naturally characterizing biological neural systems is presented in a comprehensive manner in [Levine, Brown, Shirey 2000]. Different oscillation frequencies are connected with the way in which different areas of the brain perform their functions of processing the information received from the sensors. Low level control structures must quickly react to new information received from the environment while higher level structures slowly evaluate the results associated with the present behavior policy. This is reflected in the fact that motor control occurs at approximately 200 Hz, while theta rhythms in the limbic system, where reinforcement learning is believed to operate, occur at 4–10 Hz.

In ADP, it was shown that having only a little information about the system states measured from the sensors, and extracted from the system only at specific time values (i.e., $(x(t), x(t+T), \rho(t{:}t+T))$), the Critic is able to evaluate the infinite horizon continuous-time performance of the system associated with a given control policy described in terms of the Actor parameters. The critic learns the cost function associated with a certain control behavior based on a computed temporal difference (TD) error signal (90).

It is interesting to mention here that in a number of reports, e.g., [Schultz 2004], [Schultz et al. 1997], [Doya et al. 2001], it is argued that the temporal difference error between the received and the expected rewards is physically encoded in the dopamine signal produced by basal ganglia structures in the mammal brain. At the same time, it is known that the dopamine signal encoding the temporal error difference favors the learning process by increasing the synaptic plasticity of certain groups of neurons. We also note an interesting point presented in [Perlovsky 2009] associating certain emotions with the need for cognition, i.e., emotions play the role of reinforcement signals which drive the need for cognition. This kind of reinforcement learning is located at a higher level than the dopamine driven learning, thus suggesting that there exists a hierarchy of reinforcement-based learning mechanisms in the mammal brain.

Acknowledgments

We acknowledge the support of NSF Grant ECCS-0801330 and ARO grant W91NF-05-1-0314.

Frank L. Lewis, Fellow IEEE, Fellow IFAC, Fellow U.K. Institute of Measurement & Control, PE Texas, U.K. Chartered Engineer, is Distinguished Scholar Professor and Moncrief-O'Donnell Chair at University of Texas at Arlington's Automation & Robotics Research Institute. He obtained the Bachelor's Degree in Physics/EE and the MSEE at Rice University, the MS in Aeronautical Engineering from Univ. W. Florida, and the Ph.D. at Ga. Tech. He works in feedback control, intelligent systems, and sensor networks. He is author of 6 U.S. patents, 209 journal papers, 328 conference papers, 12 books, 41 chapters, and 11 journal special issues. He received the Fulbright Research Award, NSF Research Initiation Grant, ASEE Terman Award, and Int. Neural Network Soc. Gabor Award 2008. Received Outstanding Service Award from Dallas IEEE Section, selected as Engineer of the year by Ft. Worth IEEE Section. Listed in Ft. Worth Business Press Top 200 Leaders in Manufacturing. He was appointed to the NAE Committee on Space Station in 1995. He is an elected Guest Consulting Professor at both South China University of Technology and Shanghai Jiao Tong University. Founding Member of the Board of Governors of the Mediterranean Control Association. Helped win the IEEE Control Systems Society Best Chapter Award (as

Founding Chairman of DFW Chapter), the National Sigma Xi Award for Outstanding Chapter (as President of UTA Chapter), and the US SBA Tibbets Award in 1996 (as Director of ARRI's SBIR Program).

Draguna Vrabie received her B.Sc. in 2003 and M.Sc. in 2004 from the Automatic Control and Computer Engineering Dept., "Gh. Asachi" Technical University of Iasi. Since May 2005, she is pursuing her PhD degree and is working as a research assistant at the Automation and Robotics Research Institute of the University of Texas at Arlington. Her research interests include Approximate Dynamic Programming, optimal control, adaptive control, Model Predictive Control, and general theory of nonlinear systems.

References

[1] M. Abu-Khalaf and F. L. Lewis, "Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach," *Automatica*, vol. 41, no. 5, pp. 779–791, 2005.

[2] M. Abu-Khalaf, F. L. Lewis, and J. Huang, "Policy iterations on the Hamilton-Jacobi-Isaacs equation for H-infinity state feedback control with input saturation," *IEEE Trans. Automat. Contr.*, vol. 51, no. 12, pp. 1989–1995, Dec. 2006.

[3] Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Model-free Q-learning designs for linear discrete-time zero-sum games with application to H-infinity control," *Automatica*, vol. 43, pp. 473–481, 2007.

[4] Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Discrete-time nonlinear HJB solution using approximate dynamic programming: Convergence proof," *IEEE Trans. Syst., Man, Cybern. B* (Special Issue on ADP/RL), vol. 38, no. 4, pp. 943–949, Aug. 2008.

[5] Anderson, R. M. Kretchner, P. M. Young, and D. C. Hittle, "Robust reinforcement learning control with static and dynamic stability," *Int. J. Robust Nonlinear Contr.*, vol. 11, 2001.

[6] L. Baird, "Reinforcement learning in continuous time: Advantage updating," in *Proc. Int. Conf. Neural Networks*, Orlando, FL, June 1994.

[7] S. N. Balakrishnan and V. Biega, "Adaptive critic based neural networks for aircraft optimal control," *AIAA J. Guid. Contr. Dyn.*, vol. 19, no. 4, pp. 731–739, 1996.

[8] S. N. Balakrishnan, J. Ding, and F. L. Lewis, "Issues on stability of ADP feedback controllers for dynamical systems," *IEEE Trans. Syst., Man, Cybern. B* (Special Issue on ADP/RL), vol. 38, no. 4, pp. 913–917, Aug. 2008. Invited survey paper.

[9] A. G. Barto, R. S. Sutton, and C. Anderson, "Neuron-like adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 834–846, 1983.

[10] A. G. Barto, "Connectionist learning for control," in *Neural Networks for Control*. Cambridge, MA: MIT Press, 1991.

[11] G. Barto, "Reinforcement learning and adaptive critic methods," in *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, D. A. White and D. A. Sofge, Eds. New York: Van Nostrand Reinhold, 1992, ch. 12.

[12] A. G. Barto and T. G. Dietterich, "Reinforcement learning and its relationship to supervised learning," in *Handbook of Learning and Approximate Dynamic Programming*, J. Si, A. Barto, W. Powell, and D. Wunsch, Eds. New York: Wiley-IEEE Press, 2004.

[13] R. Beard, G. Saridis, and J. Wen, "Approximate solutions to the time-invariant Hamilton-Jacobi-Bellman equation," *Automatica*, vol. 33, no. 12, pp. 2159–2177, Dec. 1997.

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

[15] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming.* MA: Athena Scientific, 1996.

[16] S. Bradtke, B. Ydstie, and A. Barto, "Adaptive linear quadratic control using policy iteration," Univ. Massachusetts, Amherst, MA, Tech. Rep. CMPSCI-94-49, June 1994.

[17] X. Cao, *Stochastic Learning and Optimization*. Berlin: Springer-Verlag, 2009.

[18] P. Dayan, "The convergence of TD(λ) for general λ," *Mach. Learn.*, vol. 8 no. 3/4, pp. 341–362, May 1992.

[19] K. Doya, "Reinforcement learning in continuous time and space," *Neural Comput.*, vol. 12, pp. 219–245, 2000.

[20] K. Doya, H. Kimura, and M. Kawato, "Neural mechanisms for learning and control," *IEEE Control Syst. Mag.*, pp. 42–54, Aug. 2001.

[21] R. Enns and J. Si, "Apache helicopter stabilization using neural dynamic programming," *AIAA J. Guid. Control Dyn.*, vol. 25, no. 1, pp. 19–25, 2002.

[22] T. Erez and W. D. Smart, "Coupling perception and action using minimax optimal control," in *Proc. ADPRL*, 2009.

[23] P. Farias, "The linear programming approach to approximate dynamic programming," in *Handbook of Learning and Approximate Dynamic Programming*, J. Si, A. Barto, W. Powell, and D. Wunsch, Eds. New York: Wiley-IEEE Press, Aug. 2004, ch. 6.

[24] L. Feldkamp and D. Prokhorov, "Recurrent neural networks for state estimation," in *Proc. 12th Yale Workshop Adaptive and Learning Systems*, New Haven, CT, 2003, pp. 17–22.

[25] S. Ferrari and R. Stengel, "An adaptive critic global controller," in *Proc. American Control Conf.*, Anchorage, AK, 2002, pp. 2665–2670.

[26] S. Hagen and B. Kröse, "Linear quadratic regulation using reinforcement learning," in *Proc. 8th Belgian–Dutch Conf. Machine Learning*, F. Verdenius and W. van den Broek, Eds. Oct. 1998, pp. 39–46.

[27] T. Hanselmann, L. Noakes, and A. Zaknich, "Continuous-time adaptive critics," *IEEE Trans. Neural Networks*, vol. 18, no. 3, pp. 631–647, 2007.

[28] P. He and S. Jagannathan, "Reinforcement learning neural-network-based controller for nonlinear discrete-time systems with input constraints," *IEEE Trans. Syst., Man, Cybern. B*, vol. 37, no. 2, pp. 425–436, Apr. 2007.

[29] G. A. Hewer, "An iterative technique for the computation of steady state gains for the discrete optimal regulator," *IEEE Trans. Automat. Contr.*, pp. 382–384, 1971.

[30] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Netw.*, vol. 3, pp. 551–560, 1990.

[31] R. Howard, *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.

[32] H. Javaherian, D. Liu, and O. Kovalenko, "Automotive engine torque and air–fuel ratio control using dual heuristic dynamic programming," in *Proc. IJCNN*, 2006, pp. 518–525.

[33] M. Kawato, "Computational schemes and neural network models for formation and control of multijoint arm trajectory," in *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, 1991, pp. 197–228.

[34] L. Kleinman, "On an iterative technique for Riccati equation computations," *IEEE Trans. Automat. Contr.*, vol. AC-13, no. 1, pp. 114–115, Feb. 1968.

[35] P. Lancaster and L. Rodman, *Algebraic Riccati Equations*. London, U.K.: Oxford Univ. Press, 1995.

[36] T. Landelius, "Reinforcement learning and distributed local model synthesis," Ph.D. dissertation, Linköping Univ., 1997.

[37] R. J. Leake and R. W. Liu, "Construction of suboptimal control sequences," *J. SIAM Contr.*, vol. 5, no. 1, pp. 54–63, 1967.

[38] D. S. Levine, V. R. Brown, and V. T. Shirey, Eds., *Oscillations in Neural Systems*. Mahwah, NJ, 2000.

[39] F. L. Lewis, K. Liu, and A. Yesildirek, "Neural net robot controller with guaranteed tracking performance," *IEEE Trans. Neural Networks*, vol. 6, no. 3, pp. 703–715, 1995.

[40] F. L. Lewis and V. Syrmos, *Optimal Control*, 2nd ed. New York: Wiley, 1995.

[41] F. L. Lewis, G. Lendaris, and D. Liu, "Special issue on approximate dynamic programming and reinforcement learning for feedback control," *IEEE Trans. Syst., Man, Cybern. B*, vol. 38, no. 4, Aug. 2008.

[42] Y. Li, N. Sundararajan, and P. Saratchandran, "Neuro-controller design for nonlinear fighter aircraft maneuver using fully tuned neural networks," *Automatica*, vol. 37, no. 8, pp. 1293–1301, Aug. 2001.

[43] Liu and H. D. Patiño, "A self-learning ship steering controller based on adaptive critic designs," in *Proc. IFAC Triennial World Congr.*, Beijing, China, July 1999, vol. J, pp. 367–372.

[44] D. Liu, X. Xiong, and Y. Zhang, "Action-dependent adaptive critic designs," in *Proc. INNS-IEEE Int. Joint Conf. Neural Networks*, Washington, DC, July 2001, pp. 990–995.

[45] X. Liu and S. N. Balakrishnan, "Adaptive critic based neuro-observer," in *Proc. American Control Conf.*, Arlington, VA, 2001, pp. 1616–1621.

[46] X. Liu and S. N. Balakrishnan, "Convergence analysis of adaptive critic based optimal control," in *Proc. American Control Conf.*, Chicago, IL, 2000, pp. 1929–1933.

[47] P. Mehta and S. Meyn, "Q-learning and Pontryagin's minimum principle," *Preprints*, 2009.

[48] J. M. Mendel and R. W. MacLaren, "Reinforcement learning control and pattern recognition systems," in *Adaptive, Learning, and Pattern Recognition Systems: Theory and Applications*, J. M. Mendel and K. S. Fu, Eds. New York: Academic Press, 1970, pp. 287–318.

[49] W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds., *Neural Networks for Control*. Cambridge, MA: MIT Press, 1991.

[50] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki, "Feedback-error-learning neural network for trajectory control of a robotic manipulator," *Neural Netw.*, vol. 1, pp. 251–265, 1988.

[51] J. Murray, C. Cox, G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Trans. Syst., Man, Cybern.*, vol. 32, no. 2, 2002.

[52] J. Murray, C. Cox, R. Saeks, and G. Lendaris, "Globally convergent approximate dynamic programming applied to an autolander," in *Proc. ACC*, Arlington, VA, 2001, pp. 2901–2906.

[53] R. Padhi, S. N. Balakrishnan, and T. Randolph, "Adaptive-critic based optimal neuro control synthesis for distributed parameter systems," *Automatica*, vol. 37, no. 8, pp. 1223–1234, Aug. 2001.

[54] R. Padhi, N. Unnikrishnan, X. Wang, and S. N. Balakrishnan, "A single network adaptive critic (SNAC) architecture for optimal control synthesis for a class of nonlinear systems," *Neural Netw.*, vol. 19, no. 10, pp. 1648–1660, Dec. 2006.

[55] L. Perlovsky, "Language and cognition," *Neural Netw.*, 2009.

[56] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. New York: Wiley, 2009.

[57] W. Powell and B. Van Roy, "ADP for high-dimensional resource allocation problems," in *Handbook of Learning and Approximate Dynamic Programming*, J. Si, A. Barto, W. Powell, and D. Wunsch, Eds. New York: Wiley-IEEE Press, Aug. 2004, ch. 10.

[58] D. Prokhorov and D. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Networks*, vol. 8, no. 5, Sept. 1997.

[59] D. V. Prokhorov and L. A. Feldkamp, "Analyzing for Lyapunov stability with adaptive critics," in *Proc. Int. Conf. Systems, Man, Cybernetics*, Dearborn, 1998, pp. 1658–1661.

[60] W. Schultz, "Neural coding of basic reward terms of animal learning theory, game theory, microeconomics and behavioral ecology," *Curr. Opin. Neurobiol.*, vol. 14, pp. 139–147, 2004.

[61] W. Schultz, P. Dayan, and P. R. Montague, "A neural substrate of prediction and reward," *Science*, vol. 275, pp. 1593–1599, 1997.

[62] J. Seiffertt, S. Sanyal, and D. C. Wunsch, "Hamilton-Jacobi-Bellman equations and approximate dynamic programming on time scales," *IEEE Trans. Syst., Man, Cybern. B*, vol. 38, no. 4, pp. 918–923, 2008.

[63] C.-Y. Seong and B. Widrow, "Neural dynamic optimization for control systems—Part I: Background," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 482–489, Aug. 2001.

[64] C.-Y. Seong and B. Widrow, "Neural dynamic optimization for control systems—Part II: Theory," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 490–501, Aug. 2001.

[65] C.-Y. Seong and B. Widrow, "Neural dynamic optimization for control systems—Part III: Applications," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 502–513, Aug. 2001.

[66] J. Si and Y.-T. Wang, "On-line control by association and reinforcement," *IEEE Trans. Neural Networks*, vol. 12, no. 2, pp. 264–276, Mar. 2001.

[67] J. Si, A. Barto, W. Powell, and D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming*. New York: Wiley-IEEE Press, 2004.

[68] R. S. Sutton and A. G. Barto, "A temporal-difference model of classical conditioning," in *Proc. 9th Annu. Conf. Cognitive Science Society*, 1987, pp. 355–378.

[69] R. S. Sutton and A. G. Barto, *Reinforcement Learning—An Introduction*. Cambridge, MA: MIT Press, 1998.

[70] R. Sutton, "Learning to predict by the method of temporal differences," *Mach. Learn.*, vol. 3, pp. 9–44, 1988.

[71] Y. Tassa, T. Erez, and W. Smart, "Receding horizon differential dynamic programming," in *Proc. Neural Information Processing Systems Conf.*, 2007, pp. 1465–1472.

[72] J. Tesauro, "Practical issues in temporal difference learning," *Mach. Learn.*, vol. 8, pp. 257–277.

[73] E. Todorov and M. I. Jordan, "Optimal feedback control as a theory of motor coordination," *Nat. Neurosci.*, vol. 5, no. 11, pp. 1226–1235, 2002.

[74] E. Todorov and Y. Tassa, "Iterative local dynamic programming," in *Proc. IEEE Symp. Adaptive Dynamic Programming and Reinforcement Learning*, 2009.

[75] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Trans. Automat. Contr.*, vol. 40, no. 9, pp. 1528–1538, Sept. 1995.

[76] B. Van Roy, D. P. Bertsekas, Y. Lee, and J. N. Tsitsiklis, "A neuro-dynamic programming approach to retailer inventory management," in *Proc. IEEE Conf. Decision Control*, San Diego, CA, 1997, pp. 4052–4057.

[77] K. Venayagamoorthy, R. G. Harley, and D. G. Wunsch, "Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator," *IEEE Trans. Neural Networks*, vol. 13, pp. 764–773, May 2002.

[78] K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, "Dual heuristic programming excitation neurocontrol for generators in a multimachine power system," *IEEE Trans. Ind. Applicat.*, vol. 39, no. 2, pp. 382–384, Mar./Apr. 2003.

[79] D. Vrabie, O. Pastravanu, M. Abu-Khalaf, and F. L. Lewis, "Adaptive optimal control for continuous-time linear systems based on policy iteration," *Automatica*, vol. 45, pp. 477–484, 2009.

[80] D. Vrabie and F. L. Lewis, "Neural network approach to continuous-time direct adaptive optimal control for partially-unknown nonlinear systems," *Neural Netw.*, to be published.

[81] F. Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: An introduction," *IEEE Comput. Intell. Mag.*, pp. 39–47, May 2009.

[82] C. Watkins, "Learning from delayed rewards," Ph.D. thesis, Cambridge Univ., Cambridge, England, 1989.

[83] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, 1992.

[84] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavior sciences," Ph.D. thesis, Committee Appl. Math. Harvard Univ., 1974.

[85] P. J. Werbos, "Neural networks for control and system identification," in *Proc. IEEE Conf. Decision and Control*, Tampa, FL, 1989.

[86] P. J. Werbos, "A menu of designs for reinforcement learning over time," in *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, 1991, pp. 67–95.

[87] P. J. Werbos, "Approximate dynamic programming for real-time control and neural modeling," in *Handbook of Intelligent Control*, D. A. White and D. A. Sofge, Eds. New York: Van Nostrand Reinhold, 1992.

[88] D. A. White and D. A. Sofge, Eds., *Handbook of Intelligent Control*. New York: Van Nostrand Reinhold, 1992.

[89] B. Widrow, N. Gupta, and S. Maitra, "Punish/reward: Learning with a critic in adaptive threshold systems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-3, no. 5, pp. 455–465, 1973.

[90] V. Yadav, R. Padhi, and S. N. Balakrishnan, "Robust/optimal temperature profile control of a high-speed aerospace vehicle using neural networks," *IEEE Trans. Neural Networks*, vol. 18, no. 4, pp. 1115–1128, July 2007.

[91] Q. Yang and S. Jagannathan, "Adaptive critic neural network force controller for atomic force microscope-based nanomanipulation," in *Proc. IEEE Int. Symp. Intelligent Control*, 2006, pp. 464–469.

[92] C. Zheng and S. Jagannathan, "Generalized Hamilton–Jacobi–Bellman formulation-based neural network control of affine nonlinear discrete-time systems," *IEEE Trans. Neural Networks*, vol. 19, no. 1, pp. 90–106, Jan. 2008.