May 28, 2000

Neuro–Dynamic Programming: Overview and Recent Trends

Benjamin Van Roy
Stanford University
bvr@stanford.edu

1 Introduction

In the study of decision–making, there is a dividing line between those who seek an understanding of how decisions *are made* and those who analyze how decisions *ought to be made* in the light of clear objectives. Among the former group are psychologists and economists who examine participants of physical systems in their full complexity. This often entails the consideration of both "rational" and "irrational" behavior. The latter group – those concerned with *rational decision–making* – includes engineers and management scientists who focus on the strategic behavior of sophisticated agents with definite purposes. The intent is to devise strategies that optimize certain criteria and/or meet specific demands. The problems here are well–defined and the goal is to find a "correct" way to make decisions, if one exists.

The self–contained character of rational decision problems has provided a ground for the development of much mathematical theory. Results of this work – as exemplified by previous chapters of this volume – provide an understanding of various possible models of dynamics, uncertainties, and objectives, as well as characterizations of optimal decision strategies in these settings. In cases where optimal strategies do exist, the theory is complemented by computational methods that deliver them.

In contrast to rational decision–making, there is no clear–cut mathematical theory about decisions made by participants of natural systems. Scientists are forced to propose speculative theories, and to refine their ideas through experimentation. In this context, one approach has involved the hypothesis that behavior is in some sense rational. Ideas from the study of rational decision–making are then used to characterize such behavior. In financial economics, this avenue has lead to utility and equilibrium theory. To this day, models arising from this school of economic thought – though far from perfect – are employed as mainstream interpretations of the dynamics of capital markets. The study of animal behavior presents another interesting case. Here, evolutionary theory and its popular precept – "survival of the fittest" – support the possibility that behavior to some extent concurs with that of a rational agent.

There is also room for reciprocal contributions from the study of natural systems to the science of rational decision–making. The need arises primarily due to the computational complexity of decision problems and the lack of systematic approaches for dealing with it. For example, practical problems addressed by the theory of dynamic programming can rarely be solved using dynamic programming algorithms because the computational time required for the generation of optimal strategies typically grows exponentially in the number of variables involved – a phenomenon known as the *curse of dimensionality*. This deficiency calls for an understanding of suboptimal decision–making in the presence of computational constraints. Unfortunately, no satisfactory theory has been developed to this end.

It is interesting to note that similar computational complexities arise in attempts to automate decision tasks that are naturally performed by humans or animals. The fact that biological mechanisms facilitate the efficient synthesis of adequate strategies motivates the possibility that understanding such mechanisms can inspire new and computationally feasible methodologies for strategic decision–making.

Over the past two decades, algorithms of *reinforcement learning* – originally conceived as descriptive models for phenomena observed in animal behavior – have grown out of the field of artificial intelligence and been applied to solving complex sequential decision problems. The success of reinforcement learning algorithms in solving large–scale problems has generated excitement and intrigue among operations researchers and control theorists, and much subsequent research has been devoted to understanding such methods and their potential. Developments have focused on a normative view, and to acknowledge the relative disconnect from descriptive models of animal behavior, some operations researchers and control theorists have come to refer to this area of research as *neuro–dynamic programming*, instead of *reinforcement learning*.

In this chapter, we provide a sample of recent developments and open issues at the frontier of research in neuro–dynamic programming. Our two points of focus are temporal–difference learning and actor–critic methods – two algorithmic ideas that have found greatest use in applications of neuro–dynamic programming and for which there has been significant theoretical progress in recent years. We begin, though, with three sections providing some background and perspective on the methodology and problems it may address.

2 Stochastic Control

As a problem formulation, let us consider a discrete–time dynamic system that, at each time t, takes on a state x_t and evolves according to

$$x_{t+1} = f(x_t, u_t, w_t),$$

where w_t is a disturbance and u_t is a control decision. Though more general (infinite/continuous) state spaces can be treated, to keep the exposition simple, we restrict attention to finite state, disturbance, and control spaces, denoted by S, W, and U, respectively. Each disturbance $w_t \in W$ is independently sampled from some fixed distribution.

A function $g : S \times U \mapsto \Re$ associates a reward $g(x_t, u_t)$ with a decision u_t made at state x_t. A *policy* is a mapping $\mu : S \mapsto U$ that generates state–contingent decisions. For each policy μ, we define a value function $J^\mu : S \mapsto \Re$ by

$$J^\mu(x) = \mathrm{E}\left[\sum_{t=0}^{\infty} \alpha^t g(x_t, \mu(x_t)) \middle| x_0 = x\right],$$

where $\alpha \in [0, 1)$ is a discount factor and the state sequence is generated according to $x_0 = x$ and $x_{t+1} = f(x_t, \mu(x_t), w_t)$. Each $J^\mu(x)$ can be interpreted as an assessment of long term rewards given that we start in state x and control the system using a policy μ. The optimal value function J^* is defined by

$$J^*(x) = \max_\mu J^\mu(x).$$

A standard result in dynamic programming states that any policy μ^* given by

$$\mu^*(x) = \operatorname*{argmax}_{u \in U} \mathop{\mathrm{E}}_{w} \Big[g(x, u) + \alpha J^*(f(x, u, w)) \Big],$$

where $\mathop{\mathrm{E}}_{w}[\cdot]$ denotes expectation with respect to the distribution of disturbances, is optimal in the sense that

$$J^*(x) = J^{\mu^*}(x),$$

for every state x (see, e.g., BertsekasDP [8]).

For illustrative purposes, let us provide one example of a stochastic control problem.

Example 1 *The video arcade game of Tetris can be viewed as an instance of stochastic control (we assume that the reader is familiar with this popular game). In particular, we can view the state* x_t *as an encoding of the current "wall of bricks" and the shape of the current "falling piece." The decision* u_t *identifies an orientation and horizontal position for placement of the falling piece onto the wall. Though the arcade game employs a more complicated scoring system, consider for simplicity a reward* $g(x_t, u_t)$ *equal to the number of rows eliminated by placing the piece in the position described by* u_t*. Then, a policy* μ *that maximizes the value*

$$J^\mu(x) = \mathrm{E}\left[\sum_{t=0}^{\infty} \alpha^t g(x_t, \mu(x_t)) \Big| x_0 = x\right],$$

essentially optimizes a combination of present and future row elimination, with decreasing emphasis placed on rows to be eliminated at times farther into the future.

Classical dynamic programming algorithms compute the optimal value function J^*. The result is stored in a "look–up" table with one entry $J^*(x)$ per state $x \in S$. When the need arises, the value function is used to generate optimal decisions. In particular, given a current state $x_t \in S$, a decision u_t is selected according to

$$u_t = \operatorname*{argmax}_{u \in U} \mathop{\mathrm{E}}_{w}\Big[g(x_t, u) + \alpha J^*(f(x_t, u, w))\Big].$$

3 Control of Complex Systems

Our primary interest is in the development of a methodology for the control of "complex systems." It is difficult to provide a precise definition for this term, but let us mention two characteristics that are common to such systems: an intractable state space and severe nonlinearities. Intractable state spaces preclude the use of classical dynamic programming algorithms, which compute and store one numerical value per state. At the same time, methods of traditional linear control, which are applicable even when state spaces are large, are ruled out by severe nonlinearities. To give a better feel for the types of problems we have in mind, let us provide a few examples.

1. **Call Admission and Routing**
   With rising demand in telecommunication network resources, effective management is as important as ever. Admission (deciding which calls to accept/reject) and routing (allocating links in the network to particular calls) are examples of decisions that must be made at any point in time. The objective is to make the "best" use of limited network resources. In principle, such sequential decision problems can be addressed by dynamic programming. Unfortunately, the enormous state spaces involved render dynamic programming algorithms inapplicable, and heuristic control strategies are used in lieu.

2. **Strategic Asset Allocation**
   Strategic asset allocation is the problem of distributing an investor's wealth among assets in the market in order to take on a combination of risk and expected return that best suits the investor's preferences. In general, the optimal strategy involves dynamic rebalancing of wealth among assets over time. If each asset offers a fixed rate of risk and return, and some additional simplifying assumptions are made, the only state variable is wealth, and the problem can be

solved efficiently by dynamic programming algorithms. There are even closed form solutions in cases involving certain types of investor preferences Merton92 [37]. However, in the more realistic situation involving risks and returns that fluctuate with economic conditions (see, e.g., Brennan97 [11]), economic indicators must be taken into account as state variables, and this quickly leads to an intractable state space.

3. **Supply–Chain Management**
   With today's tight vertical integration, increased production complexity, and diversification, the inventory flow within a corporation can be viewed as a complex network – called a *supply chain* – consisting of storage, production, and distribution sites. In a supply chain, raw materials and parts from external vendors are processed through several stages to produce finished goods. Finished goods are then transported to distributors, then to wholesalers, and finally retailers, before reaching customers. The goal in supply–chain management is to achieve a particular level of product availability while minimizing costs. The solution is a policy that decides how much to order or produce at various sites given the present state of the company and the operating environment. See Lee93 [32] and references therein for further discussion of this problem.

4. **Emissions Reductions**
   The threat of global warming that may result from accumulation of carbon dioxide and other "greenhouse gasses" poses a serious dilemma. In particular, cuts in emission levels bear a detrimental short–term impact on economic growth. At the same time, a depleting environment can severely hurt the economy – especially the agricultural sector – in the longer term. To complicate the matter further, scientific evidence on the relationship between emission levels and global warming is inconclusive, leading to uncertainty about the benefits of various cuts. One systematic approach to considering these conflicting goals involves the formulation of a dynamic system model that describes our understanding of economic growth and environmental science, as is done in Nordhaus94 [38]. Given such a model, the design of environmental policy amounts to dynamic programming. Unfortunately, classical algorithms are inapplicable due to the size of the state space.

5. **Semiconductor Wafer Fabrication**
   The manufacturing floor at a semiconductor wafer fabrication facility is organized into service stations, each equipped with specialized machinery. There is a single stream of jobs arriving on a production floor. Each job follows a deterministic route that revisits the same station multiple times. This leads to a scheduling problem where, at any time, each station must select a job to service such that (long term) production capacity is maximized (see, e.g., Kumar93 [31]). Such a system can be viewed as a special class of queueing networks, which are models suitable for a variety of applications in manufacturing, telecommunications, and computer systems. Optimal control of queueing networks is notoriously difficult, and this reputation is strengthened by formal characterizations of computational complexity in Papadimitriou99 [39].

For complex systems as those we have described, state spaces are intractable. This is a consequence of the "curse of dimensionality" – that is, the fact that state spaces generally grow exponentially in the number of state variables. For example, in a queueing network, every possible configuration of queues corresponds to a different state, and therefore, the number of states increases exponentially with the number of queues involved. For this reason, it is essentially impossible to compute (or even store) one value per state, as is required by classical dynamic programming algorithms.

There is an additional shortcoming of classical dynamic programming algorithms that is worth mentioning here – that the computations they carry out require use of transition probabilities. For many complex systems, such probabilities are not readily accessible. On the other hand, it is often easier to develop a simulator for the system that generates sample trajectories, as is commonly done to test performance of particular decision policies.

Neuro–dynamic programming algorithms aim at overcoming both deficiencies of classical algorithms. The curse of dimensionality is conquered through use of parameterized function approximators that approximate the value function in a spirit similar to statistical regression. At the same time, these algorithms rely on output generated by simulators, rather than explicite transition probabilities, in their computation.

4 Value Function Approximation

The intractability of state spaces calls for value function approximation. There are two important preconditions for the development of an effective approximation. First, we need to choose a parameterization $\tilde{J} : S \times \Re^K \mapsto \Re$ that yields a good approximation

$$\tilde{J}(x, r) \approx J^*(x),$$

for some setting of the parameter vector $r \in \Re^K$. In this respect, the choice of a suitable parameterization requires some practical experience or theoretical analysis that provides rough information about the shape of the function to be approximated. Second, we need algorithms for computing appropriate parameter values, such as those studied in neuro–dynamic programming.

Though more general classes of parameterizations have been used in neuro–dynamic programming, to keep the exposition simple, let us focus on linear parameterizations, which take the form

$$\tilde{J}(x, r) = \sum_{k=1}^{K} r(k)\phi_k(x),$$

where $\phi_1, \ldots, \phi_K$ are "basis functions" mapping S to $\Re$ and $r = (r(1), \ldots, r(K))'$ is a vector of scalar weights. In a spirit similar to that of statistical regression, the basis functions $\phi_1, \ldots, \phi_K$ are selected by a human user based on intuition or analysis specific to the problem at hand. One interpretation that is useful for the construction of basis functions involves viewing each function ϕ_k as a "feature" – that is, a numerical value capturing a salient characteristic of the state that may be pertinent to effective decision making. This general idea is probably best illustrated by a concrete example.

Example 2 *In our stochastic control formulation of Tetris, the state is an encoding of the current wall configuration and the current falling piece. There are clearly too many states for exact dynamic programming algorithms to be applicable. However, we may believe that most information relevant to game–playing decisions can be captured by a few intuitive features. In particular, one feature, say* ϕ_1*, may map states to the height of the wall. Another, say* ϕ_2*, could map states to a measure of "jaggedness" of the wall. A third might provide a scalar encoding of the type of the current falling* *piece (there are seven different shapes in the arcade game). Given a collection of such features, the next task is to select weights $r(1), \ldots, r(K)$ such that*

$$\sum_{k=1}^{K} r(k)\phi_k(x) \approx J^*(x),$$

for all states x. This approximation could then be used to generate a game–playing strategy. Such an approach to Tetris has been developed in [56] and [9]. In the latter reference, with 22 features, the authors are able to generate a strategy that eliminates an average of 3554 rows per game, reflecting performance comparable to that of an expert player.

5 Temporal–Difference Learning

In this section, we introduce temporal–difference learning as applied to tuning basis function weights in autonomous and controlled systems. Our presentation is not mathematically rigorous. Instead, emphasis is placed on conveying ideas and results at an intuitive level. More detailed discussions and mathematical analyses can be found in cited references.

5.1 Autonomous Systems

Let us begin by considering an autonomous process

$$x_{t+1} = f(x_t, w_t),$$

and aim at approximating a value function

$$J^*(x) = \mathrm{E}\left[\sum_{t=0}^{\infty} \alpha^t g(x_t) \Big| x_0 = x\right],$$

where $g(x)$ is a scalar reward associated with state x and $\alpha \in [0,1)$ is a discount factor. Note that this setting is equivalent to one where we are dealing with a controlled system and wish to approximate the value function J^μ corresponding to a fixed policy μ.

Let $\phi_1, \ldots, \phi_K$ be a collection of basis functions, and let $\tilde{J} : S \times \Re^K \mapsto \Re$ be defined by

$$\tilde{J}(x, r) = \sum_{k=1}^{K} r(k)\phi_k(x).$$

Suppose that we observe a sequence of states $x_0, x_1, x_2, \ldots$ and that at time t the weight vector has been set to some value r_t. We define the *temporal difference* d_t corresponding to the transition from x_t to x_{t+1} by

$$d_t = g(x_t) + \alpha\tilde{J}(x_{t+1}, r_t) - \tilde{J}(x_t, r_t).$$

Then, given an arbitrary initial weight vector r_0, the temporal-difference learning algorithm generates subsequent weight vectors according to

$$r_{t+1} = r_t + \gamma_t d_t z_t,$$

where γ_t is a scalar step size, and $z_t \in \Re^K$ is an *eligibility vector* defined by

$$z_t = \sum_{\tau=0}^{t} (\alpha\lambda)^{t-\tau} \phi(x_\tau),$$

where $\phi(x) = (\phi_1(x), \ldots, \phi_K(x))'$. The parameter λ takes on values in $[0, 1]$, and to emphasize its presence, the temporal-difference learning is often referred to as TD(λ). Note that the eligibility vectors can be recursively updated according to

$$z_{t+1} = \alpha\lambda z_t + \phi(x_{t+1}).$$

Let us provide one (heuristic) interpretation of the algorithm. Note that the temporal difference d_t can be viewed as a difference between two predictions of future rewards:

1. $\tilde{J}(x_t, r_t)$ is a prediction of $\sum_{\tau=t}^{\infty} \alpha^{\tau-t} g(x_\tau)$ given our current approximation $\tilde{J}(\cdot, r_t)$ to the value function.

2. $g(x_t) + \alpha\tilde{J}(x_{t+1}, r_t)$ is an "improved prediction" that incorporates knowledge of the reward $g(x_t)$ and the next state x_{t+1}.

Roughly speaking, the learning process tries to make predictions $\tilde{J}(x_t, r_t)$ consistent with their improved versions. Note that $\phi(x_t) = \nabla_r \tilde{J}(x_t, r)$. Consequently, when $\lambda = 0$, the update can be rewritten as

$$r_{t+1} = r_t + \gamma_t \nabla_r \tilde{J}(x_t, r_t)\Big(g(x_t) + \alpha\tilde{J}(x_{t+1}, r_t) - \tilde{J}(x_t, r_t)\Big).$$

The gradient can be viewed as providing a direction for the adjustment of r_t such that $\tilde{J}(x_t, r_t)$ moves towards the improved prediction. In the more general case of $\lambda \in [0, 1]$, the direction of the adjustment is determined by the eligibility vector $z_t = \sum_{\tau=0}^{t} (\alpha\lambda)^{t-\tau} \nabla_r \tilde{J}(x_\tau, r_t)$. Here, each gradient term in the summation corresponds to one of the previous states, and the temporal difference can be viewed as "triggering" adjustments of all previous predictions. The powers of α account for discounting effects inherent to the problem, while the powers of λ influence the "credit assignment" – that is, the amounts by which previous predictions are to be adjusted based on the current temporal difference.

A sizable literature addresses the dynamics of temporal–difference methods in the context of an autonomous process. Examples include Sutton88,Dayan92,Dayan94,Gurvits94,Schapire96,Tsitsiklis96,VanRoy98 [47, 16, 17, 23, 41, 56, 60]. The most recent of these results Tsitsiklis96,VanRoy98 [56, 60] state that, under appropriate technical conditions:

1. For any $\lambda \in [0, 1]$, there exists a vector $r^{(\lambda)}$ such that the sequence r_t generated by the algorithm converges (with probability one) to $r^{(\lambda)}$.

2. The limit of convergence $r^{(\lambda)}$ satisfies

$$\|J^* - \Phi r^{(\lambda)}\|_\pi \leq \frac{1}{\sqrt{1-\kappa^2}} \|\Pi J^* - J^*\|_\pi,$$

   where

$$\kappa = \frac{\alpha(1-\lambda)}{1-\lambda\alpha} \leq \alpha,$$

the norm $\|\cdot\|_\pi$ is defined by

$$\|J\|_\pi = \left(\sum_{x \in S} \pi(x) J^2(x) \right)^{1/2},$$

with π being the invariant distribution of the process, and the matrix Π projects onto the span of $\phi_1, \ldots, \phi_K$ with respect to $\|\cdot\|_\pi$.

These results imply that the iterates r_t converge to some $r^{(\lambda)}$. Furthermore, $\Phi r^{(\lambda)}$ provides an approximation to J^* in a sense that we will now describe. The term $\|\Pi J^* - J^*\|$ represents the error associated with the projection ΠJ^*. By the projection theorem, this error is minimal (if we are constrained to selecting approximations within the span of $\phi_1, \ldots, \phi_K$). The bound state above therefore establishes that the error associated with $\Phi r^{(\lambda)}$ is within a constant factor of the best possible.

5.2 Controlled Systems

The algorithm described in the previous section involves simulating a system and updating weights of an approximate value function based on observed state transitions. Unlike an autonomous system, a controlled system cannot be passively simulated and observed. Control decisions are required and influence the system's dynamics. In this section, we discuss extensions of temporal–difference learning to this context. The objective is to approximate the optimal value function of a controlled system.

5.2.1 Approximate Policy Iteration

A well–known result in dynamic programming is that, given a value function J^μ corresponding to a policy μ, an improved policy $\overline{\mu}$ can be defined by

$$\overline{\mu}(x) = \operatorname*{argmax}_{u \in U} \mathop{\mathrm{E}}_{w} \Big[g(x,u) + \alpha J^\mu(f(x,u,w)) \Big].$$

In particular, $J^{\overline{\mu}}(x) \geq J^\mu(x)$ for all $x \in S$. Furthermore, a sequence of policies $\{\mu_m | m = 0,1,2,\ldots\}$ initialized with some arbitrary μ_0 and updated according to

$$\mu_{m+1}(x) = \operatorname*{argmax}_{u \in U} \mathop{\mathrm{E}}_{w} \Big[g(x,u) + \alpha J^{\mu_m}(f(x,u,w)) \Big],$$

converges to an optimal policy μ^*. This iterative method for generating an optimal policy constitutes *policy iteration*, a classical dynamic programming algorithm due to Howard [26].

As with other dynamic programming algorithms, policy iteration suffers from the curse of dimensionality. In particular, each value function J^{μ_m} generated during the course of the algorithm can not be efficiently computed or stored. A possible approach to overcoming such limitations involves approximating each iterate J^{μ_m} in terms of a weighted combination of basis functions. For instance, letting $\phi_1, \ldots, \phi_K$ be a set of basis functions and letting $\tilde{J}(x,r) = \sum_{k=1}^K r(k)\phi_k(x)$, consider generating a sequence of weight vectors $r^1, r^2, \ldots$ by selecting each r^{m+1} such that

$$\tilde{J}(x, r^{m+1}) \approx J^{\tilde{\mu}_m}(x),$$

where $\tilde{\mu}_0$ is an arbitrary initial policy and for $m = 1, 2, 3, \ldots$,

$$\tilde{\mu}_m(x) = \operatorname*{argmax}_{u \in U} \operatorname*{E}_{w} \Big[g(x,u) + \alpha \tilde{J}(f(x,u,w), r^m) \Big].$$

We will refer to such an algorithm as *approximate policy iteration*.

There is one key component missing in our description of approximate policy iteration – a method for generating each iterate r^m. The possibility we have in mind is, of course, temporal–difference learning. In particular, we can apply the temporal–difference learning algorithm to the autonomous system resulting from simulation of the controlled system under a fixed policy $\tilde{\mu}_m$. (The dynamics are described by $x_{t+1} = f(x_t, \tilde{\mu}_m(x_t), w_t)$.) Initializing with $r_0^{m+1} = r^m$, the algorithm would generate a sequence of vectors $r_1^{m+1}, r_2^{m+1}, r_3^{m+1}, \ldots$ that converges. The limiting vector provides the subsequent iterate r^{m+1}.

To clarify the interplay between the two types of iterations involved in approximate policy iteration, let us note that we have nested sequences:

- An "external" sequence is given by $r^0, r^1, r^2, \ldots$
- For each $m = 1, 2, 3, \ldots$, an "internal" sequence is given by $r_0^m, r_1^m, r_2^m, \ldots$

For each m, the internal sequence is initialized with $r_0^{m+1} = r^m$ and the limit of convergence becomes the next element r^{m+1} of the external sequence.

The dynamics of approximate policy iteration are not very well understood. However a result from [10] to some extent motivates its use. The result states that, if there exists some $\epsilon > 0$ such that

$$\max_{x \in S} |(\Phi r^m)(x) - J^{\mu_{m-1}}(x)| \leq \epsilon,$$

for all m, then

$$\limsup_{m \to \infty} \max_{x \in S} |(\Phi r^m)(x) - J^*(x)| \leq \frac{2\alpha\epsilon}{(1-\alpha)^2}.$$

In other words, if each of the policy evaluations errs by no more than ϵ per component, approximate value iteration eventually comes within a constant factor of ϵ from the optimal value function.

5.2.2 Controlled TD

Any function $J : S \mapsto \Re$ can be used to generate a policy

$$\mu(x) = \operatorname*{argmax}_{u \in U} \operatorname*{E}_{w} \Big[g(x,u) + \alpha J(f(x,u,w)) \Big].$$

In this respect, one can view J as a guide for decision–making. The value functions $J^{\mu_0}, J^{\mu_1}, J^{\mu_2}, \ldots$ generated by (exact) policy iteration can then be viewed as a monotonically improving sequence of guides.

Recall that given a policy μ, the value function J^μ generates an improved policy. It seems therefore seems reasonable to hope that the approximation $\tilde{J}(\cdot, r^{m+1})$ to $J^{\tilde{\mu}_m}$ similarly generates a policy $\tilde{\mu}_{m+1}$ that improves on $\tilde{\mu}_m$. Now recall that, approximate policy iteration employs temporal–difference learning to compute r^{m+1} given r^m. This is done by simulating the system under the control policy $\tilde{\mu}_m$, initializing a sequence with $r_0^{m+1} = r^m$, and generating $r_1^{m+1}, r_2^{m+1}, r_3^{m+1}, \ldots$ according to the temporal–difference learning iteration. Since the corresponding sequence of functions

$\tilde{J}(\cdot, r^1), \tilde{J}(\cdot, r^2), \tilde{J}(\cdot, r^3), \ldots$ converges to $\tilde{J}(\cdot, r^{m+1})$, one might speculate that these intermediate functions themselves provide improving guides to decision–making, each of which can be used to control the system. This possibility motivates an alternative algorithm, which we refer to as *controlled TD*.

Controlled TD simulates a state trajectory $x_0, x_1, x_2, \ldots$ and then generates weight vectors $r_0, r_1, r_2, \ldots$. The initial state x_0 and weight vector r_0 can be arbitrary. Given a state x_t and a weight vector r_t, a decision u_t is generated according to

$$u_t = \operatorname*{argmax}_{u \in U} \operatorname*{E}_{w} \Big[g(x_t, u) + \alpha \tilde{J}(f(x_t, u, w), r_t) \Big].$$

The next state x_{t+1} is then given by

$$x_{t+1} = f(x_t, u_t, w_t).$$

Analogously with the autonomous case, let the temporal difference d_t be defined by

$$d_t = g(x_t, u_t) + \alpha \tilde{J}(x_{t+1}, r_t) - \tilde{J}(x_t, r_t).$$

Then, the weight vector is updated according to

$$r_{t+1} = r_t + \gamma_t d_t z_t,$$

where γ_t is a scalar step size and the eligibility vector $z_t \in \Re^K$ is once again defined by

$$z_t = \sum_{\tau=0}^{t} (\alpha\lambda)^{t-\tau} \phi(x_\tau).$$

There is little theory providing understanding of controlled TD. For the case of a "look–up table" representation – i.e., where we store one value per state, as is done by classical dynamic programming algorithms – existing results indicate that controlled TD with λ set to 0 converges so long as every state is visited infinitely often in the course of simulation Tsitsiklis94,Jaakkola94 [55, 27]. There are also results involving very restrictive types of parameterizations such as those arising from state aggregation Singh94,Tsitsiklis96,Gordon95 [45, 56, 22]. These establish convergence in the context of such approximators for variants of controlled TD that sample states with fixed relative frequencies.

Another special case for which fairly comprehensive results are available involves a version of controlled TD tailored for solving optimal stopping problems – a quite limited albeit practically relevant class of stochastic control problems. This theory establishes convergence of the algorithm to a unique limit that offers a desirable approximation to the value function VanRoy98,Tsitsiklis99 [60, 59].

In practice, controlled TD often suffers from getting "stuck" in "deadlock" situations. In particular, viewing the procedure in an anthropomorphic light, the state x_t constitutes an animal's operating environment and u_t is the action it takes. The action is selected based on an approximate value function $\tilde{J}(\cdot, r_t)$, and the weight vector r_t is improved based on experience. If the animal always selects actions in terms of a deterministic function of x_t and $\tilde{J}(\cdot, r_t)$, there is a possibility that only a small subset of the state space will ever visited and that the animal will never "learn" the value of states outside that region. This is related to the notion of a "self–fulfilling prophecy," whereby the unexplored is never explored because values learned from the explored do not promote further exploration. A modification that has been found to useful in practical applications involves adding "exploration noise" to the controls. One approach to this end involves selecting decisions by choosing $u_t = \overline{u}$ with probability

$$\frac{\exp\left(\left(\underset{w}{\mathrm{E}}\left[g(x_t, \overline{u}) + \alpha \tilde{J}(f(x_t, \overline{u}, w), r_t)\right]\right)/\delta\right)}{\sum_{u \in U} \exp\left(\left(\underset{w}{\mathrm{E}}\left[g(x_t, u) + \alpha \tilde{J}(f(x_t, u, w), r_t)\right]\right)/\delta\right)},$$

for some small parameter $\delta > 0$. Note that at any state, each decision is selected with positive probability upon each visit, and that as δ approaches 0, the probability that u_t is a decision that maximizes

$$\underset{w}{\mathrm{E}}\left[g(x_t, u) + \alpha \tilde{J}(f(x_t, u, w), r_t)\right],$$

becomes 1.

Recent theoretical results have pointed to an additional reason for exploration. The trajectory of weight vectors r_t generated by controlled TD can be viewed as an approximation to a trajectory of an ordinary differential equation. Limits of convergence of the algorithm correspond to stationary points of the ordinary differential equation. Some recent work has studied such stationary points, showing that – in the absence of exploration – there need not exist any stationary points [18]. This work also shows that, with the incorporation of exploration of the type described above, controlled TD is guaranteed to possess at least one stationary point. One might also hope that this stationary point is unique. However, as illustrated by an example in [18], this is not necessarily the case.

Due to the current absence of adequate theory, there is no streamlined and widely accepted version of controlled TD. Instead, there is a conglomeration of variants, and each one is parameterized by values that must be selected by a user. It is unclear which algorithms and parameter settings will work on a particular problem, and when a method does work, it is still unclear which ingredients are actually necessary for success. As a result, applications often require trial and error in a long process of parameter tweaking and experimentation.

5.2.3 Approximating the Q–Function

Given the optimal value function J^*, the generation of optimal control decisions

$$u_t = \underset{u \in U}{\operatorname{argmax}} \underset{w}{\mathrm{E}}\left[g(x_t, u) + \alpha J^*(f(x_t, u, w))\right],$$

requires computing one expectation per element of the decision space U, which requires in turn repeated evaluation of the system function f. One approach to avoiding this computation involves obtaining a "Q–function," as originally introduced by Watkins [62], which maps $S \times U$ to $\Re$ and is defined by

$$Q^*(x, u) = \underset{w}{\mathrm{E}}\left[g(x, u) + \alpha J^*(f(x, u, w))\right].$$

Given this function, optimal decisions can be computed according to

$$u_t = \underset{u \in U}{\operatorname{argmax}}\, Q^*(x_t, u),$$

which no longer involves taking expectations or evaluating the system function.

Q–learning Watkins89,Watkins92 [62, 63] is a variant of temporal–difference learning that approximates Q functions rather than value functions. The basis functions $\phi_1, \ldots, \phi_K$ now map $S \times U$ to $\Re$, and the objective is to obtain a weight vector $r = (r(1), \ldots, r(K))'$ such that

$$Q^*(x,u) \approx \tilde{Q}(x,u,r) = \sum_{k=1}^{K} r(k)\phi_k(x,u).$$

Like in controlled TD, Q–learning simulates a state trajectory $x_0, x_1, x_2, \ldots$ and then generates weight vectors $r_0, r_1, r_2, \ldots$. Given a state x_t and a weight vector r_t, a decision u_t is generated according to

$$u_t = \operatorname*{argmax}_{u \in U} \tilde{Q}(x_t, u, r_t).$$

The next state x_{t+1} is then given by

$$x_{t+1} = f(x_t, u_t, w_t).$$

The temporal difference d_t is defined by

$$d_t = g(x_t, u_t) + \alpha \tilde{Q}(x_{t+1}, u_{t+1}, r_t) - \tilde{Q}(x_t, u_t, r_t),$$

and the weight vector is updated according to

$$r_{t+1} = r_t + \gamma_t d_t z_t,$$

where γ_t is a scalar step size and the eligibility vector $z_t \in \Re^K$ is defined by

$$z_t = \sum_{\tau=0}^{t} (\alpha\lambda)^{t-\tau} \phi(x_\tau, u_\tau).$$

Like in the case of controlled TD, it is often desirable to incorporate exploration, for example, by selecting selecting decisions according to $u_t = \overline{u}$ with probability

$$\frac{\exp\left(\tilde{Q}(x_t, \overline{u}, r_t)/\delta\right)}{\sum_{u \in U} \exp\left(\tilde{Q}(x_t, u, r_t)/\delta\right)},$$

for some small parameter $\delta > 0$.

The analysis of Q–learning bears many similarities with that of controlled TD, and results that apply to one can often be generalized in a straightforward way to accommodate the other. For example, results applying to the "look–up table" case apply when $\lambda = 0$ to both controlled TD and Q–learning Tsitsiklis94,Jaakkola94 [55, 27]. Similarly, results on the relevance of exploration to the existence of stationary points for controlled TD deFarias99 [18] can also be extended to the case of Q–learning.

5.3 Relationship with Approximate Value Iteration

The classical value iteration algorithm can be described compactly in terms of the "dynamic programming operator" T, defined by

$$(TJ)(x) = E_w\left[g(x,w) + \alpha J(f(x,u,w))\right],$$

for any J. In particular, value iteration generates a sequence of functions according to $J_{k+1} = TJ_k$, each mapping states to real numbers. This sequence converges to the optimal value function J^*, which is the unique fixed point of T and can be used to generate an optimal policy.

Approximate value iteration – which dates all the way back to 1959 [7] – aims at approximating each iterate J_k by a linear combination of prespecified basis functions $\phi_1, \ldots, \phi_K$. In rough terms, iterates $\tilde{J}_k$ are generated according to $\tilde{J}(\cdot, r_{k+1}) = \Pi T \tilde{J}(\cdot, r_k)$, where Π is a projection operator that produces a function that is in the span of $\phi_1, \ldots, \phi_K$ and close to $T\tilde{J}_k$. The hope is that $\tilde{J}_k$ converges to a good approximation of J^*.

Recent work points out that the approximate value iteration need not possess fixed points [18], and therefore should not be expected to converge. In fact, even in cases where a fixed point exists, and even when the system is autonomous, the algorithm can generate a diverging sequence of weight vectors [60].

Controlled TD can be thought of as a stochastic approximation algorithm designed to converge on fixed points of approximate value iteration [60]. One advantage of controlled TD is its use of simulation to effectively bypass the need to explicitely compute projections required for approximate value iteration. But two features of controlled TD also come to the rescue where approximate value iteration can fail.

One advantage can be fully appreciated in the context of autonomous systems. In this case, through use of simulation, controlled TD visits states with relative frequencies equal to the steady–state distribution of underlying Markov chain. This effectively induces a projection onto the subspace spanned by basis functions with respect to a weighted quadratic norm, with weights given by the relative frequencies. It turns out that the use of such a projection, which is related to the dynamics of the underlying Markov chain, ensures in the autonomous case that approximate value iteration converges to a unique fixed point and that controlled TD converges to the same point. Without the use of simulation, it is generally difficult to implement approximate value iteration with such a projections. This is important since the use of alternative norms in projection can lead to divergence. Results along these lines are proved in [57, 60].

A second advantage, realized in the context of controlled systems, involves the possible introduction of exploration. Without exploration, controlled TD, and related versions of approximate value iteration need not possess fixed points [18].

5.4 Historical Notes

There is a long history behind the algorithms discussed in the preceding sections. We will attempt to provide a brief historical account of items that are particularly relevant to what we have presented.

The line of research originated in an area of artificial intelligence known as *reinforcement learning*. Temporal–difference – originally proposed by Sutton [47] – comprises a major development in this area, but draws on earlier work by Barto and Sutton [48, 5] on models for classical conditioning phenomena observed in animal behavior and by Barto, Sutton, and Anderson on "actor–critic methods," which will be further discussed in the next section. In the look–up table case, the algorithm also bears similarities with one proposed a decade earlier by Witten [69]. Another major development came with the thesis of Watkins [62], in which "Q–learning" was proposed, and the study of temporal–difference learning was integrated with classical ideas from dynamic programming and stochastic approximation theory. The work of Werbos [64, 65, 66] and Barto, Bradtke, and Singh [4] also contributed to this integration.

In addition to advancing the understanding of temporal–difference learning, the marriage with classical engineering ideas furthered the view of the algorithm as one for addressing complex engineering problems and lead to a number of applications. The practical potential was first demonstrated by Tesauro [52, 53, 54], who used a variant of controlled TD to produce a world–class Backgammon playing program. Several case studies involving problems such as channel allocation in cellular communication networks [44], elevator dispatching [14, 15], inventory management [61], and job–shop scheduling [70], followed to demonstrate additional signs of promise.

Since the completion of Watkin's thesis, there has been a growing literature involving the application of ideas from dynamic programming and stochastic approximation to the analysis of temporal–difference learning and its variants. However, the existing theory does not provide sufficient support for applications, as we will now explain. In controlled TD, approximation accuracy is limited by the choice of a parameterization. The hope, however, is that the iterative computation of parameters should lead to a good approximation relative to other possibilities allowed by this choice. Unfortunately, there is a shortage of theory that ensures desirable behavior of this kind.

6 Actors and Critics

The methods described thus far make use of a parameterized representation of the value function. An alternative that has been studied in other research communities as well as within the vein of neuro–dynamic programming involves parameterization of control policies and tuning of parameters via stochastic gradient methods (see, e.g., [21, 24, 67, 13, 35, 34]). Such methods simulate the system of interest and directly adapt the parameters of a controller as performance is observed.

A parameterized controller can be thought of as an *actor*, since it makes decisions and acts on them, thereby influencing the dynamics of the system. On the other hand, an approximate value function can be thought of as a *critic*, assessing alternative decisions to provide guidance. The interpretation as a critic may be particularly suitable in the context of approximate policy iteration, as described in Section 5.2.1. Here, given a policy, one generates an approximate value function (a critic) that provides an evaluation of each state when the system is controlled by this policy. Given feedback from this critic, one can select improved decisions (greedy decisions with respect to the value function).

A current area of active research in neuro–dynamic programming involves the combination of actors and critics, working in tandem to improve system performance. Recent results suggest a possibility that critics can accelerate the computation of gradients that are used to improve actor performance. This line of research provides an interesting interface between value function approximation methods of neuro–dynamic programming and Monte Carlo gradient estimation methods studied in operations research.

In this section, we will overview actor–critic methods and discuss some recent results. Before doing so, however, we will introduce an average reward formulation of stochastic control, involving the maximization of time–averaged rewards rather than discounted rewards. This formulation provides a more natural setting for the developments discussed in this section.

6.1 Averaged Rewards

As before, we consider a discrete–time stochastic system that, at each time t, evolves according to

$$x_{t+1} = f(x_t, u_t, w_t),$$

where x_t is a state in S, w_t is a disturbance drawn from W, and u is a decision selected from U. We will assume for convenience that for any policy $\mu : S \mapsto U$, the Markov chain following

$$x_{t+1} = f(x_t, u_t, \mu(x_t)),$$

is aperiodic and irreducible. The average reward of the system when operated by a policy μ is defined by

$$\overline{g}_\mu = \lim_{N\to\infty} \frac{1}{N} \mathrm{E}\left[\sum_{t=0}^{N} g(x_t, \mu(x_t))\right],$$

and the optimal average reward is

$$g^* = \max_\mu \overline{g}_\mu.$$

Analogous to the value functions employed in a discounted setting, for any policy μ, we define a *differential value function* by

$$J^\mu(x) = \lim_{N\to\infty} \mathrm{E}\left[\sum_{t=0}^{N} \left(g(x_t, \mu(x_t)) - \overline{g}_\mu\right) \Big| x_0 = x\right],$$

where $x_{t+1} = f(x_t, \mu(x_t), w_t)$. The optimal differential value function is defined by

$$J^*(x) = \lim_{N\to\infty} \mathrm{E}\left[\sum_{t=0}^{N} \left(g(x_t, \mu^*(x_t)) - g^*\right) \Big| x_0 = x\right],$$

where $x_{t+1} = f(x_t, \mu^*(x_t), w_t)$. Similarly with the discounted case, an optimal policy can be generated by greedy decisions with respect to J^*:

$$\mu^*(x) = \operatorname*{argmax}_{u\in U} \mathop{\mathrm{E}}_{w} \left[g(x,u) + J^*(f(x,u,w))\right].$$

There is a substantial literature generalizing temporal–difference learning and related theory to average reward formulations (see, e.g., [42, 43, 33, 51, 1, 2, 60, 58]). To provide a feel for the nature of such generalizations, let us describe a variant of temporal–difference learning that approximates the differential value function of an autonomous system. For this variant, the temporal difference d_t corresponding to a transition from x_t to x_{t+1} is taken to be

$$d_t = g(x_t) - \overline{g}_t + \tilde{J}(x_{t+1}, r_t) - \tilde{J}(x_t, r_t),$$

where $\overline{g}_t$ represents an approximation to the average reward. These approximations are updated according to

$$\overline{g}_{t+1} = (1-\gamma_t)\overline{g}_t + \gamma_t g(x_t)$$

The weights of an approximate value function are simultaneously adapted according to

$$r_{t+1} = r_t + \gamma_t d_t z_t,$$

with the eligibility vector z_t defined by

$$z_{t+1} = \sum_{\tau=0}^{t} \lambda^{t-\tau}\phi(x_\tau).$$

It is shown in [60, 58] that for any $\lambda \in [0,1)$ and under appropriate technical conditions, $\overline{g}_t$ converges to the true average reward and the weights r_t converge to values that offer an approximation to the desired differential value function.

6.2 Independent Actors

An actor is a parameterized class $\{\mu_\theta | \theta \in \Re^l\}$ of policies. If it were possible to compute gradients $\nabla_\theta \overline{g}_{\mu_\theta}$ of the performance with respect to parameter settings, one could improve performance via a gradient method. However, because the space of decisions is often discrete the gradient is not generally well–defined.

It is convenient to expand the class of policies under consideration to include those that select decisions randomly. By doing this, the discrete space of decisions can effectively be transformed into a continuous one. In particular, let us define a *stochastic policy* to be a function $\mu : S \times U \mapsto [0,1]$ with $\sum_{u \in U} \mu(x,u) = 1$ for all $x \in S$. Each $\mu(x,u)$ represents the probability with which decision u is selected when at state x.

Consider now a parameterized class $\{\mu_\theta | \theta \in \Re^l\}$ of stochastic policies for which the probabilities $\mu_\theta(x,u)$ are continuously differentiable functions of θ. Each stochastic policy μ generates an average reward $\overline{g}_\mu$, defined by

$$\overline{g}_\mu = \lim_{N \to \infty} \frac{1}{N} \mathrm{E}\left[\sum_{t=0}^{N} g(x_t, u_t)\right],$$

where the system is controlled by decisions u_t sampled at each time step according to probabilities $\mu_\theta(x_t, \cdot)$. It is well–known that there exists a deterministic policy that attains the optimal reward, and hence, $\max_\mu \overline{g}_\mu = g^*$, even when the maximum is taken over all stochastic policies. We define a differential value function J^μ associated with each stochastic policy by letting

$$J^\mu(x) = \lim_{N \to \infty} \mathrm{E}\left[\sum_{t=0}^{N} \left(g(x_t, u_t) - \overline{g}_\mu\right) \Big| x_0 = x\right].$$

It is also convenient to define Q values associated with each stochastic policy:

$$Q^\mu(x,u) = \mathop{\mathrm{E}}_{w}\left[g(x,u) - \overline{g} + J^\mu(f(x,u,w))\right].$$

If $\mu_\theta(x,u) > 0$ for all x and u, one can imagine employing a steepest ascent method of the form

$$\theta_{m+1} = \theta_m + \gamma_m \nabla_\theta \overline{g}_{\mu_{\theta_m}}.$$

Unfortunately, obtaining the gradient $\nabla_\theta \overline{g}_{\mu_\theta}$ poses a computational challenge. Monte Carlo simulation techniques, however, can generate estimates, which in turn can be employed in a stochastic gradient iteration. Such an iteration adapts the parameters according to

$$\theta_{m+1} = \theta_m + \gamma_m v_m,$$

where each v_m is an estimate of the gradient given parameters θ_m. Gradient estimation techniques have been studied extensively in the infinitesimal perturbation analysis literature Chong94,Ho91 [13, 24].

Let us discuss one stochastic gradient method, which was recently proposed by Marbach and Tsitsiklis Marbach98a,Marbach98b [35, 34]. The algorithm simulates a single endless trajectory of the system, updating the parameters of an actor upon visits to some distinguished state $\overline{x} \in S$. Let t_m be the mth time at which state $\overline{x}$ is visited. For times $t = t_m, \ldots, t_{m+1} - 1$, controls u_t are sampled at each state x_t according to probabilities $\mu_{\theta_m}(x_t, \cdot)$. The algorithm generates a sequence of estimates to the average reward according to

$$\overline{g}_{m+1} = \overline{g}_m + \gamma_m \sum_{t=t_m}^{t_{m+1}-1} (g(x_t, u_t) - \overline{g}_m).$$

Each gradient estimate is given by

$$v_m = \sum_{t=t_m}^{t_{m+1}-1} \hat{Q}_t \frac{\nabla_\theta \mu_{\theta_m}(x_t, u_t)}{\mu_{\theta_m}(x_t, u_t)},$$

where

$$\hat{Q}_t = \sum_{\tau=t}^{t_{m+1}-1} (g(x_\tau, u_\tau) - \overline{g}_m).$$

It is shown in Marbach98a,Marbach98b [35, 34] that under appropriate technical conditions, the average rewards $\overline{g}_{\mu_{\theta_m}}$ associated with the sequence of parameter vectors θ_m converges, and that

$$\lim_{m\to\infty} \nabla_\theta \overline{g}_{\mu_{\theta_m}} = 0,$$

with probability one. Hence, one should expect the asymptotic behavior of this stochastic gradient method to mimic that of its deterministic counterpart.

To provide some motivation for the structure of this algorithm and also to set the stage for integration of actors and critics in the next section, let us discuss the algorithm's relation to a characterization of the gradient provided in Marbach98a,Marbach98b [35, 34]:

$$\nabla_\theta \overline{g}_{\mu_\theta} = \sum_{x\in S, u\in U} \eta_\theta(x,u) Q^{\mu_\theta}(x,u) \frac{\nabla_\theta \mu_\theta(x,u)}{\mu_\theta(x,u)},$$

where $\eta_\theta(x,u) = \pi_\theta(x)\mu_\theta(x,u)$ and $\pi_\theta(x)$ denotes the steady–state probability of state x when the system is controlled by policy μ_θ. (Similar characterizations have also been employed in earlier work Cao97,Fu94,Jaakkola95 [12, 20, 28].) Note that if we have access to $Q^{\mu_\theta}(x,u)$ for every θ, x, and u, we could generate noisy estimates $v_m^\dagger$ according to

$$v_m^\dagger = \sum_{t=t_m}^{t_{m+1}-1} Q^{\mu_\theta}(x_t, u_t) \frac{\nabla_\theta \mu_{\theta_m}(x_t, u_t)}{\mu_{\theta_m}(x_t, u_t)},$$

and it would turn out that

$$\mathrm{E}[v_m^\dagger] = \nabla_\theta \overline{g}_{\mu_{\theta_m}}.$$

This is a consequence of the fact that state–decision probabilities $\eta_{\theta_m}(x,u)$ are equal to the relative frequencies with which state–decision pairs are sampled during a trajectory $x_{t_m}, x_{t_m+1}, \ldots, x_{t_{m+1}-1}$.

Since we do not have access to the desired values $Q^{\mu_{\theta_m}}(x_t, u_t)$, an estimate $\hat{Q}_t$ is employed in computing v_m. To see why $\hat{Q}_t$ may constitute a suitable estimate, note that if $\overline{g}_m = \overline{g}_{\theta_m}$, we have

$$\mathrm{E}\left[\hat{Q}_t | x_t, u_t\right] = Q^{\mu_{\theta_m}}(x_t, u_t) - J^{\mu_{\theta_m}}(\overline{x}).$$

It turns out that the constant term $J^{\mu_{\theta_m}}(\overline{x})$ bears no consequence on computation of the gradient, because for any θ,

$$\sum_{x\in S, u\in U} \eta_\theta(x,u) \frac{\nabla \mu_\theta(x,u)}{\mu_\theta(x,u)} \quad = \quad \sum_{x\in S, u\in U} \pi_\theta(x) \nabla_\theta \mu_\theta(x,u)$$

$$
\begin{aligned}
&= \sum_{x \in S} \pi_\theta(x) \nabla_\theta \left(\sum_{u \in S} \mu_\theta(x,u) \right) \\
&= \sum_{x \in S} \pi_\theta(x) \nabla_\theta (1) \\
&= 0,
\end{aligned}
$$

and therefore

$$\nabla_\theta \overline{g}_{\mu_\theta} = \sum_{x \in S, u \in U} \eta_{\theta_m}(x,u)(Q^{\mu_\theta}(x_t,u_t) - J^{\mu_{\theta_m}}(\overline{x})) \frac{\nabla \mu_{\theta_m}(x,u)}{\mu_{\theta_m}(x,u)}.$$

Hence,

$$v_m = \sum_{t=t_m}^{t_{m+1}-1} \hat{Q}_t \frac{\nabla_\theta \mu_{\theta_m}(x_t,u_t)}{\mu_{\theta_m}(x_t,u_t)},$$

serves as a noisy estimate of $Q^{\mu_{\theta_m}}(x_t,u_t)$ for the purpose of gradient estimation.

6.3 Using Critic Feedback

The stochastic gradient algorithm described in the previous section made use of estimates in place of $Q^\mu(x,u)$. Each estimate was generated based on a single sample trajectory from the state x to a distinguished state $\overline{x}$. High variance associated with such estimates can impede progress in stochastic gradient algorithms. In some sense, being based solely on a single sample, the estimate should not be expected to provide a close approximation to the expectation.

One possible motivation for the introduction of a critic in conjunction with an actor is as a mechanism for variance reduction in stochastic gradient algorithms. In particular, if a critic is able to offer accurate approximations of $Q^\mu(x,u)$, their use in place of single–sample estimates, may dramatically accelerate stochastic gradient methods.

As a concrete example, let us introduce an actor–critic algorithm that is similar in spirit to those proposed in [30, 50]. The algorithm involves an actor μ_θ and a critic that approximates Q–functions via a parameterization of the form

$$\tilde{Q}(x,u,r) = \sum_{k=1}^{K} r(k)\phi_k(x,u).$$

Parameters θ_t and r_t and an average reward estimate $\overline{g}_t$ are adapted during simulation of the system. At each time t, a decision u_t is sampled according to probabilities $\mu(x_t,\cdot)$, and the weight vector r_t is updated by temporal–difference learning. In particular, defining a temporal difference by

$$d_t = g(x_t) - \overline{g}_t + \tilde{Q}(x_{t+1}, r_t) - \tilde{Q}(x_t, r_t),$$

the weights at the next time step are given by

$$r_{t+1} = r_t + \gamma_t d_t z_t,$$

with the eligibility vector z_t defined by

$$z_{t+1} = \sum_{\tau=0}^{t} \lambda^{t-\tau} \phi(x_\tau, u_\tau).$$

The average reward estimate follows

$$\overline{g}_{t+1} = (1-\gamma_t)\overline{g}_t + \gamma_t g(x_t).$$

At the same time, to improve performance, the actor's parameters are adjusted according to

$$\theta_{t+1} = \theta_t + \beta_t v_t,$$

where β_t is a step size (possibly different in value from γ_t) and the noisy estimate v_t of the gradient is given by

$$v_t = \tilde{Q}(x_t, u_t, r_t)\frac{\nabla_\theta \mu_{\theta_t}(x,u)}{\mu_{\theta_t}(x,u)}.$$

One interpretation of this algorithm involves viewing the parameters of the critic as evolving much faster, and therefore converging faster, than those of the actor. In practice, this is achieved by keeping the step sizes β_t of the actor extremely small relative to α_t, the step sizes of the critic. In the extreme case, one might imagine that the parameters of the critic converge so fast that at every point in time $\tilde{Q}(x_t, u_t, r_t)$ looks to the actor as though it has already converged to an approximation of $Q^{\mu_{\theta_t}}$. We would then have

$$\nabla_\theta \overline{g}_{\mu_\theta} \approx \sum_{x\in S, u\in U} \eta(x,u)\tilde{Q}(x_t, u_t, r_t)\frac{\nabla_\theta \mu_\theta(x,u)}{\mu_\theta(x,u)},$$

and estimates of the Q values would no longer be noisy, possibly leading to a significant reduction in variance of gradient estimates.

In general, the reduction in variance brought about by using $\tilde{Q}(x_t, u_t, r_t)$ in place of a single sample estimate as was done in the previous section may come at a cost incurred by bias in the estimate. In particular, if $Q^{\mu_{\theta_t}}$ is not in the span of the basis functions $\phi_1, \ldots, \phi_K$, we should not expect to generate a good approximation. As discussed earlier in the context of critic–only methods, one might try to select basis functions based on engineering insights. In the context of the actor–critic algorithm we have described, however, there is another approach that leads to appropriate basis functions [30, 50]. To understand this approach, note that each component of the gradient

$$(\nabla_\theta \overline{g}_{\mu_\theta})_k = \sum_{x\in S, u\in U} \eta_\theta(x,u) Q^{\mu_\theta}(x,u)\frac{\partial \mu_\theta(x,u)/\partial\theta_i}{\mu_\theta(x,u)},$$

can be interpreted as an inner product between functions Q^{μ_θ} and

$$\frac{\partial \mu_\theta(\cdot,\cdot)/\partial\theta_i}{\mu_\theta(\cdot,\cdot)},$$

where the inner product is defined by

$$\langle f, h\rangle = \sum_{x\in S, u\in U} \eta_\theta(x,u) f(x,u) h(x,u),$$

for any scalar functions f and h with domain $S\times U$. Let $K = l$ (recall that l is the dimension of θ), and suppose we select basis functions

$$\phi_k(x,u) = \frac{\partial \mu_\theta(x,u)/\partial\theta_i}{\mu_\theta(x,u)},$$

for $k = 1, \ldots, K$. Then, given a projection $\bar{Q}$ of Q^{μ_θ} onto the span of the basis functions, with projection defined with respect to the inner product space under consideration, we have

$$(\nabla_\theta \overline{g}_{\mu_\theta})_k = \left\langle Q^{\mu_\theta}, \frac{\partial \mu_\theta(\cdot,\cdot)/\partial\theta_i}{\mu_\theta(\cdot,\cdot)} \right\rangle = \left\langle \overline{Q}, \frac{\partial \mu_\theta(\cdot,\cdot)/\partial\theta_i}{\mu_\theta(\cdot,\cdot)} \right\rangle.$$

This – together with the fact that temporal–difference learning does indeed approximate such a projection [57, 60] – suggests that the selection of basis functions is an appropriate one.

The approach we have described for basis function selection is appealing because it is automated. That is, given an actor with a current policy μ_{θ_t}, one can generate a small collection of basis functions that is sufficient for approximating $Q^{\mu_{\theta_t}}$ to an extent that facilitates improvement of actor performance just as the actual function $Q^{\mu_{\theta_t}}$ would. Note, however, that the selection of basis functions is contingent on the value of θ_t. As a consequence, the choice should probably change over time as θ_t evolves. Understanding the dynamics of actor–critic algorithms coupled with basis functions that change with the actor's evolving policy is an interesting open issue.

In our motivation of actor–critic algorithms, as well as existing analyses [30, 50], the critic is viewed as converging much faster than the actor. Essentially, the actor "waits" until the critic converges before computing the desired gradient. The potential speed–up brought about by the critic's evaluation hopefully reduces variance in gradient estimation, thereby speeding up the actor's convergence. However, it is not known whether the delays brought about by "waiting" for the critic to converge end up slowing down the actor's dynamics to an extent that negates potential improvements in gradient computation. A related research topic of interest involves understanding the dynamics of actor–critic systems where both actor and critic evolve on the same "time scale;" that is, where the actor does not "wait" for the critic to converge before attempting to compute gradients.

6.4 Historical Notes

Actor–critic methods have as long a history as does temporal–difference learning, and their stories are intertwined. Some of the earliest research in artificial intelligence on reinforcement learning involved interacting actors and critics, in which critics adapt according to temporal–difference learning. This includes the work of Barto, Sutton, and Anderson [6, 46]. However, algorithms with some similar ingredients had been proposed earlier on in control theory [69]. In artificial intelligence, actor–critic models were inspired in part as models of brain activity, and their role in neuroscience has been explored by Barto, Houk, and Adams [3, 25].

On the technical side, Williams and Baird [68] developed some of the earliest theoretical results about deterministic variants of actor–critic methods that employ exhaustive representations both of policies and of value functions. Again in a context involving exhaustive representations, convergence of a stochastic simulation–based method where the actor and critic operate on separate time scales was established by Borkar and Konda [29].

7 Closing Remarks

The "curse of dimensionality" can be viewed as the primary obstacle prohibiting effective solution methods for stochastic control problems. It is interesting to note that an analogous impediment arises in statistical regression. In particular, given an ability to collect data pairs of the form

$(x, J(x))$, the problem of producing an accurate approximation $\tilde{J}$ to the underlying function J becomes computationally intractable as the dimension of the domain increases. Similarly with the context of stochastic control, difficulties arise due to the curse of dimensionality. In the setting of statistical regression, a common approach to dealing with this limitation involves selecting a set of basis functions $\phi_1, \ldots, \phi_K$, collecting a set of input–output pairs $\{(x_1, J(x_1)), \ldots, (x_m, J(x_m))\}$, and using least–squares algorithm to compute weights $r(1), \ldots, r(K)$ that minimize

$$\sum_{i=1}^{m} \left(J(x_i) - \sum_{k=1}^{K} r(k)\phi_k(x_i) \right)^2 .$$

The result is an approximation of the form

$$\tilde{J}(x) = \sum_{k=1}^{K} r(k)\phi_k(x).$$

Though there is no systematic and generally applicable method for choosing basis functions, a combination of intuition, analysis, guesswork, and experimentation often leads to a useful selection. In fact, the combination of basis function selection and least–squares is a valuable tool that has met prevalent application.

The utility of least–squares statistical regression provides inspiration for neuro–dynamic programming. In particular, neuro–dynamic programming algorithms can be viewed as analogs to least–squares algorithms that are applicable to stochastic control rather than statistical regression. Given a stochastic control problem and a parameterized representation of the value function and/or policy, the intent is to compute parameters that lead to an effective approximation.

Though existing results provide a starting point, the development of streamlined methods and analyses applicable to general classes of stochastic control problems remains largely open. Our hope, however, is that the range of problems that can be addressed in such a manner will broaden with future research. A goal might be to eventually produce neuro–dynamic programming algorithms that are as useful and widely accessible in the context of stochastic control as is least–squares in the context of statistical regression.

Our brief account of ongoing research in neuro–dynamic programming is by no means exhaustive. We have chosen to focus discussion on temporal–difference learning and actor–critic methods, two research directions for which substantive results have been generated in recent years and in which many problems remain open. Among other interesting areas of neuro–dynamic programming research is the study of how problem structure and should influence choices of parameterized value functions and/or policies. One current thrust here, for example, aims at exploiting hierarchical structure of complex decision problems in defining abstractions, which often can be viewed as parameterizations of value functions and/or policies (see, e.g., [19, 40, 36] and references therein). We refer the reader to books by Bertsekas and Tsitsiklis [10] and Sutton and Barto [49] for excellent coverage of many additional topics.

References

Abounadi98 [1] J. Abounadi. *Stochastic Approximation for Non-Expansive Maps: Application to Q–Learning Algorithms.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1998.

Abounadi98a [2] J. Abounadi, D. P. Bertsekas, and V. S. Borkar. Learning Algorithms for Markov Decision Processes. Technical Report LIDS-P-2434, MIT Laboratory for Information and Decision Systems, 1998.

Barto95b [3] A. G. Barto. Adaptive Critics and the Basal Ganglia. In J. C. Houk, J. L. Davis, and D. G. Beiser, editors, *Models of Information Processing in the Basal Ganglia*, pages 215–232, Cambridge, MA, 1995. MIT Press.

Barto95 [4] A. G. Barto, S. J. Bradtke, and S. P. Singh. Real–Time Learning and Control Using Asynchronous Dynamic Programming. *Artificial Intelligence*, 72:81–138, 1995.

Barto82 [5] A. G. Barto and R. S. Sutton. Simulation of Anticipatory Responses in Classical Conditioning by a Neuron–Like Adaptive Element. *Behavioural Brain Research*, 4:221–235, 1982.

Barto83 [6] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuron–Like Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:835–846, 1983.

Bellman59 [7] R. E. Bellman and S. E. Dreyfus. Functional Approximation and Dynamic Programming. *Math. Tables and Other Aids Comp.*, 13:247–251, 1959.

BertsekasDP [8] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Bellmont, MA, 1995.

BertsIoffe [9] D. P. Bertsekas and S. Ioffe. Temporal Differences–Based Policy Iteration and Applications in Neuro–Dynamic Programming. Technical Report LIDS–P–2349, MIT Laboratory for Information and Decision Systems, 1996.

BertsekasNDP [10] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Bellmont, MA, 1996.

Brennan97 [11] M. J. Brennan, E. S. Shwartz, and R. Lagnado. Strategic Asset Allocation. *Journal of Economic Dynamics and Control*, 21:1377–1403, 1997.

Cao97 [12] X. R. Cao and H. F. Chen. Perturbation Realization, Potentials, and Sensitivity Analysis of Markov Processes. *IEEE Transactions on Automatic Control*, 42:1382–1393, 1997.

Chong94 [13] E. K. P. Chong and P. J. Ramadge. Stochastic Optimization of Regenerative Systems Using Infinitesimal Perturbation Analysis. *IEEE Transactions on Automatic Control*, 39:1400–1410, 1994.

Critesphd [14] R. H. Crites. *Large–Scale Dynamic Optimization Using Teams of Reinforcement Learning Agents*. PhD thesis, University of Massachusetts, Amherst, MA, 1996.

Crites96 [15] R. H. Crites and A. G. Barto. Improving Elevator Performance Using Reinforcement Learning. In *Advances in Neural Information Processing Systems 8*, Cambridge, MA, 1995. MIT Press.

Dayan92 [16] P. D. Dayan. The Convergence of TD(λ) for General λ. *Machine Learning*, 8:341–362, 1992.

Dayan94 [17] P. D. Dayan and T. J. Sejnowski. TD(λ) Converges with Probability 1. *Machine Learning*, 14:295–301, 1994.

deFarias99 [18] D. P. de Farias and B. Van Roy. On the Existence of Fixed Points for Approximate Value Iteration and Temporal Difference Learning. To appear in the *Journal of Optimization Theory and Applications*, 1999.

Dietterich97 [19] T. G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. to appear in the Journal of Artificial Inteligence Research, 1997.

Fu94 [20] M. C. Fu and J. Hu. Smoothed Perturbation Analysis Derivative Estimation for Markov Chains. *Operations Research Letters*, 15:241–251, 1994.

Glynn86 [21] P. W. Glynn. Stochastic Approximation for Monte–Carlo Optimization. In *Proceedings of the 1986 Winter Simulation Conference*, pages 285–289, 1986.

Gordon95 [22] G. J. Gordon. Stable Function Approximation in Dynamic Programming. Technical Report CMU-CS-95-103, Carnegie Mellon University, 1995.

Gurvits94 [23] L. Gurvits, L. J. Lin, and S. J. Hanson. Incremental Learning of Evaluation Functions for Absorbing Markov Chains: New Methods and Theorems. unpublished manuscript, 1994.

Ho91 [24] Y. C. Ho and X. R. Cao. *Perturbation Analysis of Discrete Event Systems*, volume 60. Kluwer Academic Publisher, Boston, MA, 1991.

Houk95 [25] J. C. Houk, J. L. Adams, and A. G. Barto. A Model of how the Basal Ganglia Generates and Uses Neural Signals that Predict Reinforcement. In J. C. Houk, J. L. Davis, and D. G. Beiser, editors, *Models of Information Processing in the Basal Ganglia*, pages 249–270, Cambridge, MA, 1995. MIT Press.

Howard60 [26] R. Howard. *Dynamic Programming and Markov Processes.* M.I.T. Press, Cambridge, MA, 1960.

Jaakkola94 [27] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. *Neural Computation*, 6:1185–1201, 1994.

Jaakkola95 [28] T. Jaakkola, S. P. Singh, and M. I. Jordan. Reinforcement Learning Algorithms for Partially Observable Markov Decision Problems. In *Advances in Neural Information Processing Systems 7*, pages 345–352, San Francisco, CA, 1995. Morgan Kauffman.

Konda99 [29] V. R. Konda and V. S. Borkar. Actor–Critic Like Learning Algorithms for Markov Decision Problems. *SIAM Journal of Control and Optimization*, 38(1):94–123, 1999.

Konda00 [30] V. R. Konda and J. N. Tsitsiklis. Actor–Critic Algorithms. In *Advances in Neural Information Processing Systems 12*, Cambridge, MA, 2000. MIT Press.

Kumar93 [31] P. R. Kumar. Re–Entrant Lines. *Queueing Systems: Theory and Applications*, 13:87–110, 1993.

Lee93 [32] H. L. Lee and C. Billington. Material Management in Decentralized Supply Chains. *Operations Research*, 41(5):835–847, 1993.

Mahadevan96 [33] S. Mahadevan. Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results. *Machine Learning*, 22:1–38, 1996.

Marbach98b [34] P. Marbach. *Simulation–Based Optimization of Markov Reward Processes.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1998.

Marbach98a [35] P. Marbach and J.N. Tsitsiklis. Simulation–Based Optimization of Markov Reward Processes. Submitted to the IEEE Transactions on Automatic Control, 1998.

McGovern98 [36] A. McGovern, D. Precup, S. P. Singh, and R. S. Sutton. Hierarchical Optimal Control of MDPs. In *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems*, pages 186–191, 1998.

Merton92 [37] R. C. Merton. *Continuous–Time Finance.* Basil Blackwell, Oxford, UK, 1992.

Nordhaus94 [38] W. D. Nordhaus. *Managing the Global Commons: the Economics of Climate Change.* MIT Press, Cambridge, MA, 1994.

Papadimitriou99 [39] C. H. Papadimitriou and J. N. Tsitsiklis. The Complexity of Optimal Queueing Network Control. *Mathematics of Operations Research*, 24(2):293–305, 1999.

Parr98 [40] R. E. Parr. *Hierarchical Control and Learning for Markov Decision Processes.* PhD thesis, University of California, Berkeley, Berkeley, CA, 1998.

Schapire96 [41] R. E. Schapire and M. K. Warmuth. On the Worst–Case Analysis of Temporal–Difference Learning Algorithms. *Machine Learning*, 22:95–122, 1996.

Schwartz93 [42] A. Schwartz. A Reinforcement Learning Method for Maximizing Undiscounted Rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 298–305, 1993.

Singh94b [43] S. P. Singh. Reinforcement Learning Algorithms for Average Payoff Markovian Decision Processes. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 202–207, 1994.

Singh97 [44] S. P. Singh and D. P. Bertsekas. Reinforcement Learning for Dynamic Channel Allocation in Cellular Telephone Systems. In *Advances in Neural Information Processing Systems 10*, Cambridge, MA, 1997. MIT Press.

Singh94 [45] S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement Learning with Soft State Aggregation. In *Advances in Neural Information Processing Systems 7*, Cambridge, MA, 1994. MIT Press.

Sutton84 [46] R. S. Sutton. *Temporal Credic Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, Amherst, MA, 1984.

Sutton88 [47] R. S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44, 1988.

Sutton81 [48] R. S. Sutton and A. G. Barto. Toward a Modern Theory of Adaptive Networks: Expectation and Prediction. *Psychological Review*, 88:135–170, 1981.

SuttonRL [49] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

Sutton00 [50] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems 12*, Cambridge, MA, 2000. MIT Press.

Tadepalli98 [51] P. Tadepalli and D. Ok. Model–Based Average Reward Reinforcement Learning. To appear in *Artificial Intelligence*, 1998.

Tesauro92 [52] G. J. Tesauro. Practical Issues in Temporal Difference Learning. *Machine Learning*, 8:257–277, 1992.

Tesauro94 [53] G. J. Tesauro. TD–Gammon, a Self–Teaching Backgammon Program, Achieves Master–Level Play. *Neural Computation*, 6(2):215–219, 1994.

Tesauro95 [54] G. J. Tesauro. Temporal Difference Learning and TD–Gammon. *Communications of the ACM*, 38:58–68, 1995.

Tsitsiklis94 [55] J. N. Tsitsiklis. Asynchronous Stochastic Approximation and Q–Learning. *Machine Learning*, 16:185–202, 1994.

Tsitsiklis96 [56] J. N. Tsitsiklis and B. Van Roy. Feature–Based Methods for Large Scale Dynamic Programming. *Machine Learning*, 22:59–94, 1996.

Tsitsiklis97a [57] J. N. Tsitsiklis and B. Van Roy. An Analysis of Temporal–Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

Tsitsiklis99a [58] J. N. Tsitsiklis and B. Van Roy. Average–Cost Temporal–Difference Learning. *Automatica*, 35(11):1799–1808, 1999.

Tsitsiklis99 [59] J. N. Tsitsiklis and B. Van Roy. Optimal Stopping of Markov Processes: Hilbert Space Theory, Approximation Algorithms, and an Application to Pricing High-Dimensional Financial Derivatives. *IEEE Transactions on Automatic Control*, 44(10):1840–1851, 1999.

VanRoy98 [60] B. Van Roy. *Learning and Value Function Approximation in Complex Decision Processes.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1998.

VanRoy97a [61] B. Van Roy, D. P. Bertsekas, Y. Lee, and J. N. Tsitsiklis. A Neuro–Dynamic Programming Approach to Retailer Inventory Management. In *Proceedings of the IEEE Transactions on Automatic Control*, 1997.

Watkins89 [62] C. J. C. H. Watkins. *Learning From Delayed Rewards.* PhD thesis, Cambridge University, Cambridge, UK, 1989.

Watkins92 [63] C. J. C. H. Watkins and P. Dayan. Q–learning. *Machine Learning*, 8:279–292, 1992.

Werbos87 [64] P. J. Werbos. Building and Understanding Adaptive Systems: a Statistical/Numerical Approach to Factory Automation and Brain Research. *IEEE Transactions on Systems, Man, and Cybernetics*, 17:7–20, 1987.

Werbos92a [65] P. J. Werbos. Approximate Dynamic Programming for Real–Time Control and Neural Modeling. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control*, 1992.

Werbos92b [66] P. J. Werbos. Neurocontrol and Supervised Learning: An Overview and Evaluation. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control*, 1992.

Williams92 [67] R. J. Williams. Simple Statistical Gradient Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

Williams93 [68] R. J. Williams and L. C. Baird. Analysis of Some Incremental Versions of Policy Iteration: First Steps Toward Understanding Actor–Critic Learning Systems. Technical Report NU–CCS–93–11, College of Computer science, Northeastern University, 1993.

Witten77 [69] I. H. Witten. An Adaptive Optimal Controller for Discrete–Time Markov Environments. *Information and Control*, 34:286–295, 1977.

Zhang95 [70] W. Zhang and T. G. Dietterich. A Reinforcement Learning Approach to Job Shop Scheduling. In *Proceeding of the IJCAI*, 1995.