REINFORCEMENT LEARNING AND DYNAMIC PROGRAMMING

Andrew G. Barto*

**University of Massachusetts, Department of Computer Science, Amherst, MA 01003, USA*

Abstract. Reinforcement learning refers to a class of learning tasks and algorithms based on experimental psychology's principle of reinforcement. Recent research uses the framework of stochastic optimal control to model problems in which a learning agent has to incrementally approximate an optimal control rule, or policy, often starting with incomplete information about the dynamics of its environment. Although these problems have been studied intensively for many years, the methods being developed by reinforcement learning researchers are adding some novel elements to classical dynamic programming solution methods. This article provides a brief account of these methods, explains what is novel about them, and suggests what their advantages might be over classical applications of dynamic programming to large–scale stochastic optimal control problems.

Key Words. Adaptive control, dynamic programming, function approximation, large scale systems, learning control, Monte Carlo method, optimal control, stochastic approximation, stochastic control.

1. INTRODUCTION

The term reinforcement comes from studies of animal learning in experimental psychology, where it refers to the occurrence of an event, in the proper relation to a response, that tends to increase the probability that the response will occur again in the same situation (Kimble, 1961). Although the specific term "reinforcement learning" is not used by psychologists, it has been widely adopted by theorists in engineering and artificial intelligence to refer to a class of learning tasks and algorithms based on this principle of reinforcement (e.g., (Minsky, 1961; Mendel and McLaren, 1970)). A reinforcement learning problem is usually formulated mathematically as an optimization problem with the objective of finding an action, or a policy for producing actions, that is optimal according to a given objective function.

Receiving the most attention in recent research are reinforcement learning problems in which the objective function measures the learning agent's behavior as it extends over time. In these problems, the relevant consequences of an action may not be available immediately after that action is taken. Recent research models these problems as stochastic optimal control problems in which the controller (the learning agent) has to find (or, more realistically, has to approximate) an optimal feedback control rule (a policy for acting), usually starting with incomplete information about the dynamics of the controlled system (the agent's environment). Although these problems have been studied intensively for many years, the methods being developed by reinforcement learning researchers are adding some novel elements to classical dynamic programming (DP) solution methods. Specifically, reinforcement learning algorithms based on DP all involve *interleaving* steps related to the operations of various DP algorithms with steps of real or simulated control. In other words, a DP algorithm is not used stictly as an off–line algorithm for designing a policy before that policy is used for control. Instead, DP acts as the organizing principle for incrementally compiling information acquired during real or simulated control experience.

This article provides a brief account of these methods, explains what is novel about them, and suggests what their advantages might be over classical applications of DP to large–scale stochastic optimal control problems. This account, which has been assembled by many researchers over roughly the last five years, represents our current state of understanding rather than the intuition underlying the origination of these methods. Indeed, DP–based learning originated at least as far back as Samuel's famous checkers player of the 1950s (Samuel, 1967; Samuel, 1959), which made no explicit reference to the DP literature existing at that time. Perhaps the earliest connection between learning and DP was suggested in 1977 by Werbos (1977), and much of the current interest is attributable to Watkins' 1989 dissertation (Watkins, 1989). Additional information on this class of algorithms can be found in (Barto, 1992; Barto *et al.*, 1994; Sutton, 1992; Werbos,

1992).

We begin with a brief description of the conventional DP approaches to these problems, restricting attention to finite Markov decision processes (MDPs), a formalism on which most of the rigorous theory of reinforcement learning relies. The principles of reinforcement learning, however, are by no means restricted to this class of problems.

2. THE CONVENTIONAL APPROACH

2.1. *Finite Markov Decision Processes*

At each discrete time step a controller observes a system's state x, contained in a finite set X, and selects a control action a from a finite set A_x. A reward with expected value $R(x,a)$ is delivered, and the state at the next time step is y with probability $p^a(x,y)$. A (stationary) policy $\pi : X \rightarrow \cup_{x \in X} A_x$ specifies that the controller executes action $\pi(x) \in A_x$ whenever it observes state x. For any policy π and $x \in X$, let $V^\pi(x)$ denote the *expected infinite-horizon discounted return* from x given that the controller uses policy π. Letting r_t denote the reward at time t, this is defined as:

$$V^\pi(x) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | x_0 = x \right], \qquad (1)$$

where x_0 is the system's initial state, γ, $0 \leq \gamma < 1$, is a factor used to discount future rewards, and E_π is the expectation assuming the controller always uses policy π. It is usual to call $V^\pi(x)$ the *value* of x under π. The function V^π is the *value function* corresponding to π.

This is a finite, infinite–horizon, discounted MDP. The objective is to find an optimal policy, i.e., a policy, π^*, that maximizes the value of each state x defined by (1). The unique *optimal value function*, V^*, is the value function corresponding to any optimal policy. Additional details on this and other types of MDPs can be found in many references (e.g., (Bertsekas, 1987; Ross, 1983)).

2.2. *Dynamic Programming*

In the absence of structure beyond that described above, DP provides the only exact solution methods for MDPs short of an exhaustive search of policy space. *Value iteration* is the DP algorithm that successively approximates V^* as follows. At each iteration k, it updates an approximation V_k of V^* by applying the following operation for each state x:

$$V_{k+1}(x) = \max_{a \in A_x} [R(x,a) + \gamma \sum_{y \in X} p^a(x,y) V_k(y)]. \qquad (2)$$

We call this operation a "backup" because it updates a state's value by transferring to it information about the approximate values of its possible successor states. Applying this backup operation once to each state is often called a *sweep*. Starting with an arbitrary initial function V_0, the sequence $\{V_k\}$ produced by repeated sweeps converges to V^*. When V^* is approximated to a desired accuracy, an optimal policy is taken to be any policy that for each x selects an action that realizes the maximum on the right–hand side of (2).

In the asynchronous version of value iteration (Bertsekas, 1982; Bertsekas, 1983), a backup can be applied at any time to any state using whatever values happen to be available for the successor states. As long as the value of each state continues to be backed up, this also converges to V^*. Thus, systematic sweeps of the state set are not necessary.

Policy iteration produces a sequence of policies that converges to an optimal policy. It starts with a an arbitrary initial policy, π_0, and computes its value function, V^{π_0}, then defines a new policy, π_1, that is optimal with respect to V^{π_0}. The process repeats starting with π_1. Each policy generated is either an improvement over the preceding policy, or is an optimal policy, π^*. Policy iteration's computational bottleneck is the necessity to evaluate a policy at each step, which requires solving a system of $|X|$ linear equations.

Although DP algorithms are polynomial in the number of states, they can be impractical because many problems give rise to very large state sets. For example, if X is a discretization of a multi–dimensional continuous space, its size is an exponential function of the dimension (Bellman's "curse of dimensionality"). Performing even a single value iteration sweep, or a single policy evaluation step of policy iteration, is often impractical. In fact, to exactly compute a backup operation (2) for a single state can itself be impractical due to the required sum over X and/or the max over A_x. It is also clear that these algorithms require a complete model of the MDP in the form of knowledge of the transition probabilities, $p^a(x,y)$, and the reward expectations, $R(x,a)$, $x, y \in X$, $a \in A$.

2.3. *The Case of Incomplete Information*

In the absence of a complete and accurate model of the MDP (the case of incomplete information), perhaps the most common approach (see, e.g., (Kumar and Varaiya, 1986; Kumar, 1985)) is to try to identify the unknown system and execute a DP algorithm under the assumption that the system model is correct (the certainty equivalence approach). This leads to the class of indirect

adaptive control methods which at each time step of on–line interaction with the unknown system: 1) udpate the system model using current observations, 2) run a DP algorithm using the latest model, and 3) select a control action on the basis of the DP algorithm's results, taking into account the necessity to maintain behavioral variety (to "explore") due to the conflict between identification and control.

This approach thus builds conservatively upon the results for the complete information case, producing methods whose main attraction is theoretical tractability: they are practical only for very small problems. It is often not feasible to execute a DP algorithm *once*, let alone at each time step of control. Mitigating this somewhat is the fact that the number of iterations of, say, policy iteration, tends to be small if the initial policy is the final policy produced at the preceding time step. However, even in this case, at least one policy evaluation step is required, which can be too time–consuming to accomplish before the next control decision must be made, and can often be too time–consuming to accomplish under any circumstances.

3. APPROXIMATING DYNAMIC PROGRAMMING

The computational difficulties of solving large MDPs, with complete or incomplete information, are well–known, and many methods have been proposed for approximating their solutions with less effort (e.g., (Jacobson and Mayne, 1970; Norman, 1972)). To the best of this author's knowledge, DP–based reinforcement learning methods are novel in their combination of Monte Carlo, stochastic approximation, and function approximation techniques. Specifically, these algorithms combine some, or all, of the following features:

1. Avoid exhaustive sweeps by restricting computation to states on, or in the neighborhood of, multiple sample trajectories, either real or simulated (the Monte Carlo aspect).
2. Simplify the basic DP backup by sampling: for example, estimate the expected values on the right–hand side of (2) by sampling from the appropriate distributions (the stochastic approximation aspect).
3. Represent value functions and/or policies more compactly than lookup–table representations by using function approximation methods, such as neural networks.

In what follows, we elaborate each of these properties by describing several reinforcement learning algorithms.

3.1. *Avoiding Exhaustive Sweeps*

The reinforcement learning algorithm closest to conventional DP is *real–time dynamic programming* (RTDP) (Barto *et al.*, 1994). Of the three properties listed above, it has only property 1. RTDP is the result of using sample trajectories—generated by simulation or by the actual control process—to determine the states to which value iteration backups (2) are applied. The backups are applied along these trajectories. This can be thought of as executing asynchronous value iteration (asynchronous because it does not use systematic sweeps) concurrently with a real or simulated control process. The sample trajectory determines the states to which the backup operator is applied, and the current estimate of V^* determines the selection of control actions. The simplest case applies the backup operator only to the states visited in the sample trajectories, but it can be useful to apply the backup operator to neighboring states as well. Instead of performing exhaustive sweeps, RTDP thus attempts to focus its computation on regions of the state set that are likely to be relevant in actual control.

RTDP is most useful when control is constrained to begin in a designated set of initial states. In such cases, an optimal policy need only be defined for states reachable under optimal control from the initial set. Of course, this region is unknown without knowledge of an optimal policy. RTDP can be understood as an attempt to successively confine value iteration to this region. Barto *et al.* (1994) give conditions under which RTDP is guaranteed to converge to a policy whose restriction to this region is optimal in stochastic shortest path problems.

Note that even when applied during actual control, RTDP requires a model of the MDP since it uses the value iteration backup (2). An adaptive version of RTDP (Barto *et al.*, 1994) applicable to the incomplete information case is analogous to the conventional indirect adaptive methods described above. It ties to identify the system and uses its latest model to provide the information necessary to perform backups. Unlike conventional indirect methods, however, this version of RTDP does not execute a complete value iteration computation at each time step, and thus can be applied to problems with very large state sets. RTDP is related to the *Learning–Real–Time A** algorithm of Korf (1990), the *Dyna* architecture of Sutton (1990), the *prioritized sweeping* algorithm of Moore and Atkeson (1993), and to the method proposed by Jalali and Ferguson (1989).

3.2. *Simplifying Backups*

An algorithm called *Q-learning* proposed by Watkins (1989) dispenses with the value iteration backup in favor of an operation that 1) usually requires much less computation, and 2) does not require a model of the MDP. Instead of updating estimates of $V^*(x)$, it updates estimates of optimal action–values $Q^*(x,a)$, which give the expected return for executing $a \in A_x$ when observing x and following an optimal policy thereafter. Let Q_k denote the estimate of Q^* at iteration k. The Q–Learning backup for (x,a) is

$$\begin{aligned} Q_{k+1}(x,a) &= (1-\alpha_k)Q_k(x,a) \\ &+ \alpha_k[r + \gamma \max_{b \in A_y} Q_k(y,b)], \end{aligned} \tag{3}$$

where r is a sample from the reward process with mean $R(x,a)$, y is a sample from the state–transition distribution $p^a(x,\cdot)$, and α_k is a relaxation parameter.

Using a lookup–table to store the values $Q_k(x,a)$, if the Q–learning backup is applied to each state–action pair infinitely often, and some additional natural conditions hold, then Q–learning converges to Q^* with probability one. The Q–learning backup is usually applied along sample trajectories in a manner similar to RTDP. In contrast to the value iteration backup used by RTDP, however, the Q–learning backup does not require a model of the MDP because the observed reward and next state can respectively serve as the samples r and y. They can be sampled on–line from reward and state–transition distributions of the actual system, which can can remain unknown. Additionally, the actions that are optimal with respect to an estimate of Q^* can be determined without a model: for state x, such an action is given by $\arg\max_{a \in A_x} Q_k(x,a)$.

Q–learning is a stochastic approximation of value iteration. By maintaining values for state–action pairs instead of just actions, Q–learning effectively performs updates that are unbiased estimates of value iteration backups. The convergence of Q–learning has been proven from several different perspectives. Watkins and Dayan (1992) provided the first complete proof, and more recent proofs are based on extensions of the theory of stochastic approximation (Jaakkola *et al.*, to appear; Tsitsiklis, 1993). When a single action is available for each state ($|A_x| = 1$, for all $x \in X$), Q–learning reduces to TD(0), one of Sutton's *temporal difference* algorithms (1988) which is applicable to the policy evaluation problem. Bradtke (1994) recently showed how to extend Q–learning to a recursive least squares formulation.

3.3. *Function Approximation*

Backup equations such as (2) and (3) easily can be converted into rules for updating a parameterized approximation of V_k or Q_k. For example, Q_k might be represented as $Q_k(x,a) = f(\theta_k, \phi_x, a)$, where θ_k is the parameter vector at iteration k, ϕ_x is a feature vector representating state x, and f is a given real–valued function differentiable with respect to θ_k for all (ϕ_x, a). Then the backup (3) yields the following gradient–descent parameter update rule:

$$\begin{aligned} \theta_{k+1} &= \theta_k + \\ \alpha_k[r &+ \gamma \max_{b \in A_y} Q_k(y,b) - Q_k(x,a)] \nabla_{\theta_k} Q_k(x,a) \end{aligned} \tag{4}$$

where $\nabla_{\theta_k} Q_k(x,a)$ denotes the gradient vector at (ϕ_x, a) of f as a function of θ_k. However, rules like this present the least understood aspect of DP–based reinforcement learning. It is easy to generate examples showing that these rules can be unstable, or can converge to the wrong function, even under conditions that would seem to be very favorable (e.g., (Bradtke, 1994)). On the other hand, there are also examples in which this approach works very well.

A remarkable demonstration of this, as well as other properties of DP–based reinforcement learning algoritms, is provided by Tesauro's backgammon playing program called *TD–Gammon* (Tesauro, 1992; Tesauro, 1994). Starting with little backgammon knowledge beyond the rules of the game, this program has learned to play near the level of the world's strongest grandmasters (and is still improving). Backgammon can be formulated as a finite MDP whose states are possible states of the game (board configurations plus other information) and whose actions are moves of pieces. Transitions are stochastic because on each turn, a player tosses a pair of die to determine a set of possible moves, and the opponent's responding move can be viewed as an additional stochastic factor. The reward is always zero unless the program wins, in which case it is 1. The value function corresponging to a policy, then, assigns to each state the probability that the program will win from that state using the given policy. The optimal value function assigns the winning probability for optimal play.

Given assumptions about the opponent, this MDP can in principle be solved using DP. However, since there are approximately 10^{20} states, and about 400 possible moves at each play, even a single sweep of value iteration is out of the question (it would take more than 1K years using a 1K MIPS processor), as is a single policy evaluation or policy improvement step of policy iteration. Instead, TD–Gammon played repeated

games against an opponent (which happened to be itself). The games were simulated, which is possible because this is an MDP with complete information. A game is a sample trajectory of the MDP.

TD–Gammon maintains and updates an estimate of V^*, which it represents using a multi–layer neural network whose input vector consists of features describing the state of the game, some of which are defined using expert knowledge of backgammon. At each play, it selects the action (move) that leads to the most promising successor state as evaluated by its current estimate of V^*. Then using the estimated value of the actual next state, it updates the network parameters (its weights) by error–backpropagation (Rumelhart *et al.*, 1986) based on a gradient–descent TD rule similar to (4). High–level play was obtained after about 500K games, which took on the order of months of computer time. While this is a significant amount of computation, it is inconsequential compared to what a conventional DP method would require.

4. CONCLUSION

Although the reasons for TD–Gammon's success are still somewhat mysterious, this system illustrates the potential of DP–based reinforcement learning methods for finding useful approximate solutions to large–scale stochastic optimal control problems. Because computation is guided by simulated games, TD–Gammon takes advantage of the fact that very many states have a very low probability of occurring in real games. In this respect, it is similar to other Monte Carlo methods in automatically allocating computation in proportion to that computation's influence on the desired result (Barto and Duff, 1994). TD–Gammon also illustrates that DP–based reinforcement learning can require relatively little computation per–time–step of real or simulated control. The methods that use backups based on reward and next–state samples, instead of the full backups of conventional DP, share the property of stochastic approximation methods in being applicable on–line during actual control when there is no model of the system. Finally, parameterized function approximation, although not well–understood at present, can be used to produce algorithms whose space complexity is within the realm of practicality. Current research is seeking to exploit these properties in a range of large–scale problems.

5. ACKNOWLEDGEMENTS

This research was supported in part by grants from the National Science Foundation (ECS-9214866) and the Air Force Office of Scientific Research, Bolling AFB (AFOSR F49620-93-1-0269).

6. REFERENCES

Barto, A. and M. Duff (1994). Monte Carlo matrix inversion and reinforcement learning. In: *Advances in Neural Information Processing Systems 6* (J. D. Cowan, G. Tesauro and J. Alspector, Eds.). Morgan Kaufmann. San Francisco. pp. 687–694.

Barto, A. G., S. J. Bradtke and S. P. Singh (1994). Learning to act using real-time dynamic programming. *Artificial Intelligence*. In press.

Barto, A.G. (1992). Reinforcement learning and adaptive critic methods. In: *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches* (D. A. White and D. A. Sofge, Eds.). pp. 469–491. Van Nostrand Reinhold. New York.

Bertsekas, D. P. (1982). Distributed dynamic programming. *IEEE Transactions on Automatic Control* **27**, 610–616.

Bertsekas, D. P. (1983). Distributed asynchronous computation of fixed points. *Mathematical Programming* **27**, 107–120.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall. Englewood Cliffs, NJ.

Bradtke, S. J. (1994). Incremental Dynamic Programming for On-Line Adaptive Optimal Control. PhD thesis. University of Massachusetts, Computer Science Dept. Technical Report 94-62.

Jaakkola, T., M. I. Jordan and S. P. Singh (to appear). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*.

Jacobson, D. H. and D. Q. Mayne (1970). *Differential Dynamic Programming*. Elsevier. New York.

Jalali, A. and M. Ferguson (1989). Computationally efficient adaptive control algorithms for markov chains. In: *Proceedings of the 28th Conference on Decision and Control*. Tampa, Florida. pp. 1283–1288.

Kimble, G. A. (1961). *Hilgard and Marquis' Contitioning and Learning*. Appleton-Century-Crofts. New York.

Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence* **42**, 189–211.

Kumar, P. R. (1985). A survey of some results in stochastic adaptive control. *SIAM Journal of Control and Optimization* **23**, 329–380.

Kumar, P. R. and P. Varaiya (1986). *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice-Hall. Englewood Cliffs, NJ.

Mendel, J. M. and R. W. McLaren (1970). Rein-

forcement learning control and pattern recognition systems. In: *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications* (J. M. Mendel and K. S. Fu, Eds.). pp. 287–318. Academic Press. New York.

Minsky, M. L. (1961). Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers* **49**, 8–30. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*. McGraw-Hill, New York, 406–450, 1963.

Moore, A. W. and C. G. Atkeson (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* **13**(1), 103–130.

Norman, J. M. (1972). *Heuristic Procedures in Dynamic Programming*. Manchester University Press. Manchester, UK.

Ross, S. (1983). *Introduction to Stochastic Dynamic Programming*. Academic Press. New York.

Rumelhart, D. E., G. E. Hinton and R. J. Williams (1986). Learning internal representations by error propagation. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations* (D. E. Rumelhart and J. L. McClelland, Eds.). Bradford Books/MIT Press. Cambridge, MA.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development* pp. 210–229. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York, 1963.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. II—Recent progress. *IBM Journal on Research and Development* pp. 601–617.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning* **3**, 9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: *Proceedings of the Seventh International Conference on Machine Learning*. Morgan Kaufmann. San Mateo, CA. pp. 216–224.

Sutton, R. S., Ed.) (1992). *A Special Issue of Machine Learning on Reinforcement Learning*. Vol. 8. *Machine Learning*. Also published as *Reinforcement Learning*, Kluwer Academic Press, Boston, MA 1992.

Tesauro, G. J. (1992). Practical issues in temporal difference learning. *Machine Learning* **8**, 257–277.

Tesauro, G. J. (1994). Temporal difference learning in TD-Gammon. *Communications of the ACM*. in press.

Tsitsiklis, J. N. (1993). Asynchronous stochastic approximation and Q-learning. Technical Report LIDS-P-2172. Laboratory for Information and Decision Systems, MIT. Cambridge, MA.

Watkins, C. J. C. H. (1989). Learning from Delayed Rewards. PhD thesis. Cambridge University. Cambridge, England.

Watkins, C. J. C. H. and P. Dayan (1992). Q-learning. *Machine Learning* **8**, 279–292.

Werbos, P. J. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook* **22**, 25–38.

Werbos, P. J. (1992). Approximate dynamic programming for real-time control and neural modeling. In: *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches* (D. A. White and D. A. Sofge, Eds.). pp. 493–525. Van Nostrand Reinhold. New York.