Incremental Multi-Step Q-Learning

Jing Peng
College of Computer Science
Northeastern University
Boston, MA 02115
jp@ccs.neu.edu

Ronald J. Williams
College of Computer Science
Northeastern University
Boston, MA 02115
rjw@ccs.neu.edu

Abstract

This paper presents a novel incremental algorithm that combines Q-learning, a well-known dynamic programming-based reinforcement learning method, with the TD(λ) return estimation process, which is typically used in actor-critic learning, another well-known dynamic programming-based reinforcement learning method. The parameter λ is used to distribute credit throughout sequences of actions, leading to faster learning and also helping to alleviate the non-Markovian effect of coarse state-space quantization. The resulting algorithm, *Q(λ)-learning*, thus combines some of the best features of the Q-learning and actor-critic learning paradigms. The behavior of this algorithm is demonstrated through computer simulations of the standard benchmark control problem of learning to balance a pole on a cart.

1 INTRODUCTION

The incremental multi-step Q-learning (Q(λ)-learning) method is a new direct (or model-free) algorithm that extends the one-step Q-learning algorithm (Watkins 1989) by combining it with TD(λ) returns for general λ (Sutton 1989) in a natural way for delayed reinforcement learning. By allowing corrections to be made incrementally to the predictions of observations occurring in the past, the Q(λ)-learning method propagates information rapidly to where it is important. The Q(λ)-learning algorithm works significantly better than the one-step Q-learning algorithm on a number of tasks and its basis in the integration of one-step Q-learning and TD(λ) returns makes it possible to take advantage of some of the best features of the Q-learning and actor-critic learning paradigms and to bridge the gap between them. It can also serve as a basis for developing various multiple time scale learning mechanisms that are essential for applications of reinforcement learning to real world problems.

2 TD(λ) RETURNS

Direct dynamic programming-based reinforcement learning algorithms are based on updating state values or state-action values according to state transitions as they are experienced. Each such update is in turn based on the use of a particular choice of estimator for the value being updated. This section describes an important and computationally useful class of such estimators – the TD(λ) estimators (Sutton 1988, Watkins 1989).

Let the world state at time step t be x_t, and assume that the learning system then chooses action a_t. The immediate result is that a reward r_t is received by the learner and the world undergoes a transition to the next state, x_{t+1}. The objective of the learner is to choose actions maximizing discounted cumulative rewards over time. More precisely, let γ be a specified discount factor in $[0, 1)$. The *total discounted return* (or simply *return*) received by the learner starting at time t is given by

$$\mathbf{r}_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n} + \cdots.$$

The objective is to find a policy π, or rule for selecting actions, so that the expected value of the return is maximized. It is sufficient to restrict attention to policies that select actions based only on the current state (called *stationary* policies). For any such policy π and for any state x we define

$$V^\pi(x) = E[\mathbf{r}_0 | x_0 = x, a_i = \pi(x_i) \text{ for all } i \geq 0],$$

the expected total discounted return received when starting in state x and following policy π thereafter. If π is an optimal policy we also use the notation V^* for V^π. Many dynamic programming-based reinforcement learning methods involve trying to estimate the state values $V^*(x)$ or $V^\pi(x)$ for a fixed policy π.

An important class of methods for estimating V^π for a given policy π is the TD(λ) estimators, which have

been investigated by Sutton (Sutton 1984,1988) and later by Watkins (Watkins 1989). Following Watkins' notation, let $\mathbf{r}_t^{(n)}$ denote the *corrected n-step truncated return* for time t, given by

$$\begin{aligned}\mathbf{r}_t^{(n)} &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \\ &\quad \gamma^{n-1} r_{t+n-1} + \gamma^n \hat{V}_{t+n}^{\pi}(x_{t+n}) \qquad (1)\end{aligned}$$

where $\hat{V}_t^{\pi}$ is the estimate of V^{π} at time t. If $\hat{V}^{\pi}$ were equal to V^{π}, then the corrected truncated returns would be unbiased estimators of V^{π}. Watkins (Watkins 1989) shows that corrected truncated returns have the *error-reduction property* in that the expected value of the corrected truncated return is closer to V^{π} than $\hat{V}^{\pi}$ is. Then Sutton's TD(λ) return from time t is

$$\begin{aligned}\mathbf{r}_t^{\lambda} &= (1-\lambda)[\mathbf{r}_t^{(1)} + \lambda \mathbf{r}_t^{(2)} + \lambda^2 \mathbf{r}_t^{(3)} + \cdots] \\ &= (1-\lambda)(r_t + \gamma \hat{V}_t^{\pi}(x_{t+1})) + \\ &\quad (1-\lambda)\lambda(r_t + \gamma r_{t+1} + \gamma^2 \hat{V}_{t+1}^{\pi}(x_{t+2})) + \cdots \\ &= r_t + \gamma(1-\lambda)\hat{V}_t^{\pi}(x_{t+1}) + \\ &\quad \gamma\lambda[r_{t+1} + \gamma(1-\lambda)\hat{V}_{t+1}^{\pi}(x_{t+2}) + \cdots\end{aligned}$$

Intuitively, $\mathbf{r}^{\lambda}$ is a weighted average of corrected truncated returns in which the weight of $\mathbf{r}^{(n)}$ is proportional to λ^n, where $0 < \lambda < 1$. As a result, $\mathbf{r}^{\lambda}$ has the error-reduction property. In fact, it is shown (Dayan 1992, Sutton 1988) that under certain conditions the expected value of $\mathbf{r}^{\lambda}$ converges to V^{π}.

The TD(λ) return can also be written recursively as

$$\mathbf{r}_t^{\lambda} = r_t + \gamma(1-\lambda)\hat{V}_t^{\pi}(x_{t+1}) + \gamma\lambda \mathbf{r}_{t+1}^{\lambda} \qquad (2)$$

Then the TD(0) return is just

$$\mathbf{r}_t^0 = r_t + \gamma \hat{V}_t^{\pi}(x_{t+1})$$

and the TD(1) return is

$$\mathbf{r}_t^1 = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

which is the exact actual return. Watkins argues that, in a Markov decision problem, the choice of λ is a trade-off between bias and variance. Sutton's empirical demonstration (Sutton 1988) favors intermediate values of λ that are closer to 0. For further details, see (Sutton 1988, Watkins 1989).

3 ONE-STEP Q-LEARNING

One-step Q-learning of Watkins (Watkins 1989), or simply Q-learning, is a simple incremental algorithm developed from the theory of dynamic programming (Ross 1983) for delayed reinforcement learning. In Q-learning, policies and the value function are represented by a two-dimensional lookup table indexed by state-action pairs. Formally, using notation consistent with that of the previous section, for each state x and action a let

$$\begin{aligned}Q^*(x,a) &= E\{r_0 + \gamma V^*(x_1) | x_0 = x, a_0 = a\} \qquad (3) \\ &= R(x,a) + \gamma \sum_y P_{xy}(a) V^*(y) \qquad (4)\end{aligned}$$

where $R(x,a) = E\{r_0 | x_0 = x, a_0 = a\}$, and $P_{xy}(a)$ is the probability of reaching state y as a result of taking action a in state x. It follows that

$$V^*(x) = \max_a Q^*(x,a) \qquad (5)$$

Intuitively, Equation (4) says that the state-action value, $Q^*(x,a)$, is the expected total discounted return resulting from taking action a in state x and continuing with the optimal policy thereafter. More generally, the Q function can be defined with respect to an arbitrary policy π as

$$Q^{\pi}(x,a) = R(x,a) + \gamma \sum_y P_{xy}(a) V^{\pi}(y), \qquad (6)$$

and Q^* is just Q^{π} for an optimal policy π.

The Q-learning algorithm works by maintaining an estimate of the Q^* function, which we denote by $\hat{Q}^*$, and adjusting $\hat{Q}^*$ values (often just called *Q-values*) based on actions taken and reward received. This is done using Sutton's prediction difference, or TD error (Sutton 1988) - the difference between the immediate reward received plus the discounted value of the next state and the Q-value of the current state-action pair:

$$r + \gamma \hat{V}^*(y) - \hat{Q}^*(x,a) \qquad (7)$$

where r is the immediate reward, y is the next state resulting from taking action a in state x, and $\hat{V}^*(x) = \max_a \hat{Q}^*(x,a)$. Then the values of $\hat{Q}^*$ are adjusted according to

$$\hat{Q}^*(x,a) = (1-\alpha)\hat{Q}^*(x,a) + \alpha(r + \gamma \hat{V}^*(y)) \qquad (8)$$

where $\alpha \in (0,1]$ is a learning rate parameter. In terms of the notation described in the previous section, Equation (8) may be rewritten as

$$\hat{Q}^*(x,a) = (1-\alpha)\hat{Q}^*(x,a) + \alpha \mathbf{r}^0 \qquad (9)$$

That is, the Q-learning method uses TD(0) as its estimator of expected returns.

Note that the current estimate of the Q^* function implicitly defines a greedy policy by $\pi(x) = \arg\max_a \hat{Q}^*(x,a)$. That is, the greedy policy is to select actions with the largest estimated Q-value.

It is important to note that the one-step Q-learning method does not specify what actions the agent should take at each state as it updates its estimates. In fact, the agent may take whatever actions it pleases. This means that Q-learning allows arbitrary experimentation while at the same time preserving the current best

estimate of states' values. This is possible because Q-learning constructs a value function on the state-action space, instead of the state space. It constructs a value function on the state space only indirectly. Furthermore, since this function is updated according to the ostensibly optimal choice of action at the following state, it does not matter what action is actually followed at that state. For this reason, the estimated returns in Q-learning are not contaminated by "experimental" actions (Watkins 1989), so Q-learning is not *experimentation-sensitive.*

On the other hand, because actor-critic learning updates the state value at any state based on the actual action selected, not on what would have been the optimal choice of action, it is experimentation-sensitive. In fact, in an actor-critic learning system, experimental actions always affect the agent's estimated returns, potentially hampering the agent's learning efficiency, as we shall see in Section 5.

To find the optimal Q function eventually, however, the agent must try out each action in every state many times. It has been shown (Watkins 1989, Watkins & Dayan 1992) that if Equation (8) is repeatedly applied to all state-action pairs in any order in which each state-action pair's Q-value is updated infinitely often, then $\hat{Q}^*$ will converge to Q^* and $\hat{V}^*$ will converge to V^* with probability 1 as long as α is reduced to 0 at a suitable rate.

Finally, Watkins (Watkins 1989) has also described possible extensions to the one-step Q-learning method by using different value estimators, such as $\mathbf{r}^\lambda$ for $0 < \lambda < 1$, and he has illustrated the use of $\mathbf{r}^\lambda$ returns in Q-learning in his empirical demonstrations by memorizing past experiences and calculating these returns at the end of each learning period, where a learning period specifies the number of experiences occurring in the past the agent needs to store. The following section derives a novel algorithm that enables the value estimation process to be done incrementally.

4 Q(λ)-LEARNING

This section derives the Q(λ)-learning algorithm combining TD(λ) returns for general λ with Q-learning in an incremental way. Note that in terms of the notation introduced here, one-step Q-learning is simply Q(0)-learning, making it a special case.

For simplicity, in what follows we drop the superscript π in V^π and assume that the given policy π is the agent's greedy policy. Now let

$$e_t = r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t) \tag{10}$$

and

$$e'_t = r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{Q}_t(x_t, a_t) \tag{11}$$

where $\hat{V}(x) = \arg\max_a \hat{Q}(x, a)$. Then, if we use Equation (11) for one step and Equation (10) thereafter, the difference between the TD(λ) return of Equation (2) and the estimated Q-value can be written as

$$\begin{aligned}
\mathbf{r}_t^\lambda - \hat{Q}_t(x_t, a_t) = & \\
& \{r_t + \gamma(1-\lambda)\hat{V}_t(x_{t+1}) + \\
& \gamma\lambda[r_{t+1} + \gamma(1-\lambda)\hat{V}_{t+1}(x_{t+2}) + \\
& \gamma\lambda[r_{t+2} + \gamma(1-\lambda)\hat{V}_{t+2}(x_{t+3}) + \cdots\} \\
& -\hat{Q}_t(x_t, a_t) \\
= & \{r_t + \gamma\hat{V}_t(x_{t+1}) - \hat{Q}_t(x_t, a_t) + \\
& \hat{Q}_t(x_t, a_t) - \gamma\lambda\hat{V}_t(x_{t+1}) + \\
& \gamma\lambda[r_{t+1} + \gamma\hat{V}_{t+1}(x_{t+2}) - \hat{V}_{t+1}(x_{t+1}) + \\
& \hat{V}_{t+1}(x_{t+1}) - \gamma\lambda\hat{V}_{t+1}(x_{t+2}) + \\
& \gamma\lambda[r_{t+2} + \cdots\} - \hat{Q}_t(x_t, a_t)
\end{aligned}$$

Thus,

$$\begin{aligned}
\mathbf{r}_t^\lambda - \hat{Q}_t(x_t, a_t) = & \\
& e'_t + \gamma\lambda e_{t+1} + \gamma^2\lambda^2 e_{t+2} + \gamma^3\lambda^3 e_{t+3} + \cdots \\
& \sum_{n=1}^{\infty} (\gamma\lambda)^n [\hat{V}_{t+n}(x_{t+n}) - \hat{V}_{t+n-1}(x_{t+n})].
\end{aligned} \tag{12}$$

If the learning rate is small, so that Q is adjusted slowly, then the second summation on the right-hand side of the above equation will be small.

1. $\hat{Q}(x, a) = 0$ and $Tr(x, a) = 0$ for all x and a
2. Do Forever:
   (a) $x_t \leftarrow$ the current state
   (b) Choose an action a_t that maximizes $\hat{Q}(x_t, a)$ over all a
   (c) Carry out action a_t in the world. Let the short term reward be r_t, and the new state be x_{t+1}
   (d) $e'_t = r_t + \gamma\hat{V}_t(x_{t+1}) - \hat{Q}_t(x_t, a_t)$
   (e) $e_t = r_t + \gamma\hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)$
   (f) For each state-action pair (x, a) do
   - $Tr(x, a) = \gamma\lambda Tr(x, a)$
   - $\hat{Q}_{t+1}(x, a) = \hat{Q}_t(x, a) + \alpha Tr(x, a) e_t$

   (g) $\hat{Q}_{t+1}(x_t, a_t) = \hat{Q}_{t+1}(x_t, a_t) + \alpha e'_t$
   (h) $Tr(x_t, a_t) = Tr(x_t, a_t) + 1$

Figure 1: The Q(λ)-Learning Algorithm.

The $Q(\lambda)$-learning algorithm is summarized in Figure 1, where $Tr(x, a)$ is the "activity" trace of state-action pair (x, a), corresponding to the "eligibility" trace as described in (Barto, Sutton & Anderson 1983). This description does not specify how actions are chosen during experimentation. One sensible approach that

we used in the simulation experiments to be described below is to choose actions randomly with a bias toward the ostensibly optimal action. In particular, one can choose actions randomly according to the Boltzmann distribution, with the probability of choosing any action a in state x proportional to

$$\exp(\hat{Q}(x,a)/T)$$

where T is a temperature parameter (set equal to 1 in the experiments described below). Choosing an action probabilistically ensures the necessary exploration of different actions.

The main difficulty associated with Q(λ)-learning in a Markov decision process is that rewards received after a non-greedy action cannot be used to evaluate the agent's greedy policy since this will not be the policy that was actually followed. In other words, Q(λ)-learning is experimentation-sensitive, assuming that $\lambda > 0$ is fixed. For a discussion of some ways around this difficulty, see (Watkins 1989).

Still another difficulty is that changes in $\hat{Q}$ at each time step may affect $\mathbf{r}^\lambda$, which will in turn affect $\hat{Q}$, and so on. However, these effects may not be significant for small α since they are proportional to α^2 (Peng 1993).

At each time step, the Q(λ)-learning algorithm loops through a set of state-action pairs which grow linearly with time. This can cause serious concerns on a serial machine, since, in the worst case, the algorithm may have to enumerate the entire state-action space. However, the number of state-action pairs for which actual updating is required can be kept at a manageable level by maintaining only those state-action pairs whose activity trace $(\gamma\lambda)^n$ is significant, since this quantity declines exponentially when $\gamma\lambda < 1$. Another approach is to implement a Q(λ)-learning system on a parallel machine in which each state-action pair is mapped onto a separate processor. This corresponds directly to the kind of neural network implementation first envisioned for the actor-critic approach (Barto, Sutton & Anderson). A possible alternative way to use a parallel machine, appropriate for situations when much of the state-action space may never be experienced, is to dynamically allocate processors to only those state-action pairs actually experienced. This is particularly appealing if the system can exhibit more focused behaviors.

Finally, it is interesting to note that both Q(λ)-learning and actor-critic learning (Sutton 1984) use TD(λ) returns as their value estimators through a trace mechanism. Therefore, it seems reasonable to expect the Q(λ)-learning algorithm to exhibit beneficial performance characteristics attributable to the use of TD(λ) returns for $\lambda > 0$, as illustrated in (Barto, Sutton & Anderson 1983, Sutton 1988). At the same time, both Q(λ)-learning and one-step Q-learning construct a value function on the state-action space rather than just the state space, making them both capable of discriminating between the effects of choosing different actions in each state. Thus, while Q(λ)-learning is experimentation-sensitive, unlike one-step Q-learning, it seems reasonable to expect it to be less so than actor-critic learning. Overall, then, Q(λ)-learning appears to incorporate some of the best features of the Q-learning and actor-critic learning paradigms into a single mechanism. Furthermore, it can be viewed as a potential bridge between them. The following section will illustrate this still further.

5 EXPERIMENTAL DEMONSTRATION

This section describes an experimental result evaluating the performance of the Q(λ)-learning method described above in the familiar domain of the classic cart-pole problem. As a comparison, the actor-critic learning system (Barto, Sutton & Anderson 1983) and the one step Q-learning system (Watkins 1989) are also implemented. These algorithms were chosen for comparison because they are all direct (or model-free) approaches.

5.1 CART-POLE PROBLEM

Figure 2 illustrates the cart-pole problem. The cart is free to travel left or right along a one dimensional bounded track. The pole is hinged to the top of the cart and is free to move in the vertical plane aligned with the track. The objective is to learn to push the cart left or right so as to keep the pole balanced more or less vertically above the cart, and also to keep the cart from colliding with the ends of track.

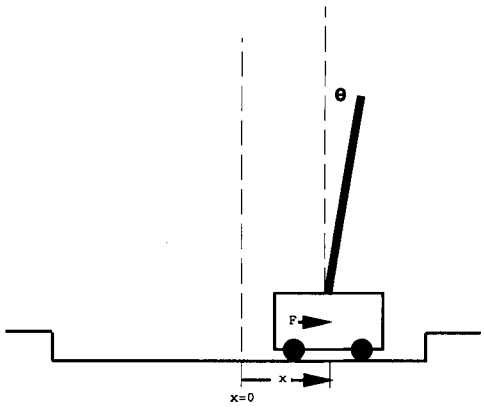

Figure 2: The Cart-Pole System

One can pose this as a learning problem in several ways. The formulation of the learning problem adopted here is the same as that described in (Barto, Sutton & Anderson 1983). That is, the learner has access to the state vector $(x, \dot{x}.\theta, \dot{\theta})$ at each time

step and can select one of two actions, a rightward or leftward force on the cart. The cart-pole system begins with $\theta = 0$, $\dot{\theta} = 0$, $x = 0$, and $\dot{x} = 0$. If the pole falls over more than 12 degrees from vertical, or if the cart hits the track boundary, a failure is said to occur. All immediate rewards are zero except upon failure, when a reward of negative one is delivered. When a failure occurs, the cart-pole system is reset to the initial state, and a new attempt to balance the pole begins. A *trial* is a complete balancing attempt.

5.2 SIMULATION RESULTS

The representation used in all experiments reported here coincides with that used in (Barto, Sutton & Anderson 1983), with the state space quantized coarsely into 162 cells. For the actor-critic experiments, then, there were 162 actor entries (representing the policy in each cell) and 162 critic values, while the Q(λ)-learning algorithm and the one-step Q-learning algorithm each required maintaining $162 \times 2 = 324$ numbers. Euler's method with a time step of 0.02 second was used to approximate numerically the solution of the equations of motion of the cart-pole system given in (Barto, Sutton & Anderson 1983).

Table 1: Summary of Simulation Results on the Cart-Pole Problem

ALGORITHMS	NO. EXPS	NO. TRIALS
Q-learning	142171	478
Actor-Critic learning	461583	108
Q(λ)-learning	30571	95

A series of runs of each learning method attempting to control the cart-pole system were carried out, where each run consisted of a sequence of trials. All table entries of each method were set to zero at the beginning of each run. Also, all the trace variables were set to zero at the start of each trial. Except for the initial conditions of the random number generator, identical parameter values were used for all runs. Each run consisted of a number of trials until the cart-pole system remained balanced for more than 100000 time steps (approximately 35 minutes of simulated real time), in which case the run was terminated. The balancing duration of the system was measured by interspersing test trials, with learning turned off, with the normal activities of the learning agent.

Table 1 shows the simulation results of the Q-learning algorithm, the actor-critic learning algorithm, and the Q(λ)-learning algorithm. The numbers of Table 1 are averages of performance over ten runs. For the implementation of the actor-critic learning system, the identical parameter values published in (Barto, Sutton & Anderson 1983) were used. For the Q-learning system, $\alpha = 0.2$ learning rate value was picked that seemed to give the best result. A slightly different version of the Q(λ)-learning algorithm of Figure 1 was simulated in the experiment, in which the learning rate parameter for the current state-action pair and the one for the state-action pairs occurring in the past were allowed to be independent. The parameter values used in producing the Q(λ)-learning results of Table 1 were $\alpha = 0.2$ (the learning rate for the current state-action pair), $\beta = 0.1$ (the learning rate for the state-action pairs occurring in the past), and $\lambda = 0.9$. These values were chosen since they seemed to give the best performance among other values tried. The discount factor, γ, was set to 0.95 for all three methods. Figure 3 shows the simulation results of the three learning systems in terms of the number of actual experiences and the number of trials, respectively, averaged over 10 runs.

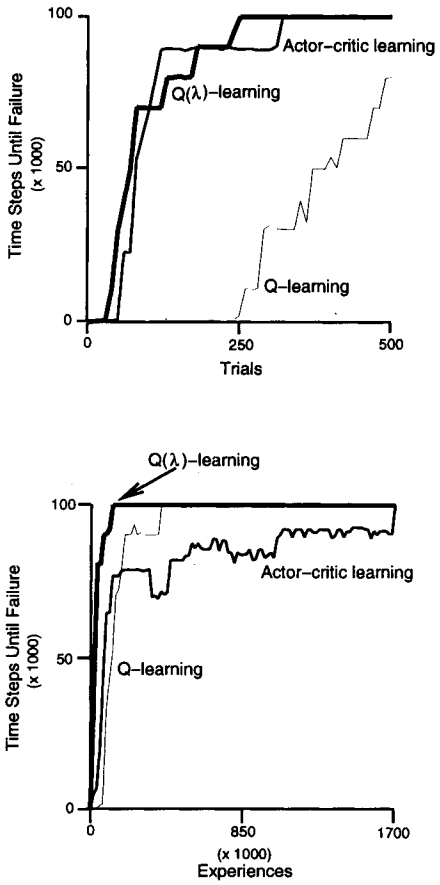

Figure 3: Performance Of Three Learning Systems, Measured Against The Number Of Trials (top) And The Number Of Experiences (bottom).

The Q(λ)-learning system achieved much better results in terms of both the number of actual experiences, where an experience is a state transition occurring in the world, and the number of trials than did the Q-learning system. As Table 1 shows, it took, on average, 30571 actual experiences (approximately 10 min-

utes of simulated real time) for the Q(λ)-learning algorithm to balance the pole for more than 100000 time steps, whereas the Q-learning algorithm took about 48 minutes. Furthermore, the Q(λ)-learning system tended to solve the problem before it had experienced 100 failures. In contrast, the Q-learning system experienced a much greater number of failures before it learned to solve the problem.

The significant performance improvement of the Q(λ)-learning system over the simple Q-learning system is clearly due to the use of the TD(λ) return estimation process, which has the effect of making alterations to past predictions throughout each trial. One natural explanation for the improved learning efficiency in this case is that TD(λ) propagates information backward much faster than does TD(0). For every action taken in the world, TD(λ) provides some degree of updating for essentially all recent predictions, while TD(0) only updates the most recent one. This is the reason that $\lambda = 0.3$ was found empirically to give optimal performance in the random-walk prediction problem studied by Sutton (1988). If this is the main benefit conferred by TD(λ), one might expect model-based, multiple-update methods like priority-Dyna (Peng 1993, Peng & Williams 1993, Moore & Atkeson 1994), to perform at least as well. However, additional experiments, whose details we omit here, were carried out using such techniques and found to give results significantly worse than those obtained using Q(λ)-learning. We believe the reason for this is that the coarse state-space quantization used here has the effect of making the environment non-Markovian, and increasing λ makes TD(λ) less sensitive to this non-Markovian effect.

The Q(λ)-learning system performed slightly better than the actor-critic learning system in terms of the number of trials, but significantly better in terms of the number of actual experiences. As Table 1 indicates, it took, on average, about 2.5 hours of simulated real time for the actor-critic learning system to learn to solve the problem, as compared with 10 minutes for the Q(λ)-learning system. The reason for this may have to do with the following. Although both Q(λ)-learning and actor-critic learning use TD(λ) returns as their estimators, the ways in which the TD(λ) returns are computed in the two systems are different. Q(λ)-learning computes the TD(λ) return estimates using the current best estimated action values in the states actually visited, while actor-critic learning computes the TD(λ) returns using essentially an average of values of actions taken so far. This means that Q(λ)-learning is less experimentation-sensitive than is actor-critic learning. We believe that the efficient use of experience measures in general the degree of accuracy to which the estimated returns are computed, and thus is critical for any learning system to be successful in more realistic environments.

A close look at Figure 3 reveals that Q(λ)-learning has a strikingly similar performance characteristic with actor-critic learning in terms of the number of trials and a very similar performance characteristic with Q-learning in terms of the number of actual experiences. That is, Q(λ)-learning has achieved the best of both the actor-critic and Q-learning paradigms, which is what it was designed to do.

Obviously, we cannot make general statements about the relative merit of these systems based solely on these experiments. Nevertheless, the argument made by Werbos (Werbos 1990) may help explain this to a certain extent.

It should be pointed out that both the fixed period learning process of Watkins (Watkins 1989) for sufficiently long learning periods and the action-replay process of Lin (Lin 1992) produce the same beneficial effects as that of Q(λ)-learning. However, both of these approaches operate in "batch" mode in that they replay, backwards, the memorized sequence of experiences that the learning agent has recently had.

6 CONCLUSION

The Q(λ)-learning algorithm is of interest because of its incrementality and its relationship to Q-learning (Watkins 1989) and actor-critic learning (Barto, Sutton & Anderson 1983, Sutton 1984). The experiments reported here demonstrate that the Q(λ)-learning algorithm inherits the best qualities of both the actor-critic learning algorithm and the Q-learning algorithm. However, this algorithm, unlike the one-step Q-learning algorithm, cannot be expected to converge to the correct Q^* values under an arbitrary policy that tries every action in every state (although the obvious strategies of gradually reducing λ or gradually turning down the Boltzmann temperature as learning proceeds would probably allow such convergence). In spite of this, the Q(λ)-learning algorithm has always outperformed the one-step Q-learning algorithm on all the problems we have experimented with so far. Furthermore, this also means that an analytic understanding of this algorithm will necessarily require more complex arguments than are used to prove convergence of one-step Q-learning.

From a computational standpoint, the incrementality of Q(λ)-learning makes it more attractive than Watkins' batch mode learning and Lin's action replay process since the computation can be distributed over time more evenly, and thus under many circumstances can ease overall demands on the memory and speed. Similar arguments are made in (Sutton 1988). One additional pleasing characteristic of the Q(λ)-learning method is that it achieves greater computational efficiency without having to learn and use a model of the world (Peng 1993, Peng & Williams 1993, Sutton 1990) and is well suited to parallel implementation.

This paper has only examined the Q(λ)-learning algorithm in which the TD(λ) returns are computed by taking the maximum Q values at each state visited. There are other possibilities, however. For example, the algorithm may estimate the TD(λ) returns by using a fixed Q value at each state. This amounts to estimating Q^π for a particular policy π. What is interesting for this is that it can be shown (Peng 1993) that for a given policy π, Q(λ)-learning converges with probability one to the correct Q^π values under appropriate conditions. Also see (Jaakkola, Jordan & Singh 1993).

Finally, it is clear that in continuous-time systems, or even systems where time is discrete but very fine-grained, the use of algorithms that propagate information back one step at a time can make no sense or at least be of little value. In these cases the use of TD(λ) methods is not a luxury but a necessity. In general, λ can be viewed as a time scale parameter in such situations, and we argue that better understanding of its use in this regard is an important area for future research.

Acknowledgements

We wish to thank Rich Sutton for his many valuable suggestions and continuing encouragement. This work was supported by Grant IRI-8921275 from the National Science Foundation.

References

Barto, A. G., Sutton, R. S. & Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* **13**:835-846.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning* **8**:117-138.

Jaakkola, T., Jordan, M. I. & Singh, S. P. (1993). On the convergence of stochastic iterative dynamic programming algorithms. *ML93 Reinforcement Learning Workshop.*

Lin, L. J. (1992). *Reinforcement learning for robots using neural networks.* Ph. D. Dissertation, Carnegie Mellon University, PA.

Moore, A. W. & Atkeson, C. G. (1994). Prioritized sweeping: reinforcement learning with less data and less time. *Machine Learning* **13**(1):103-130.

Peng, J. (1993). *Efficient Dynamic Programming-Based Learning for Control.* Ph. D. Dissertation, Northeastern University, Boston, MA 02115.

Peng, J. & Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behavior* **1**(4):437-454.

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

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,* 216-224.

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

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning.* Ph. D. Dissertation, University of Massachusetts, Amherst (also COINS Technical Report 84-02).

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

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* Ph. D. Dissertation, King's College, UK.

Werbos, P. J. (1990). Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks* **3**:179-189.