Open Theoretical Questions in Reinforcement Learning

Richard S. Sutton

AT&T Labs, Florham Park, NJ 07932, USA,
`sutton@research.att.com`, `www.cs.umass.edu/~rich`

Reinforcement learning (RL) concerns the problem of a learning agent interacting with its environment to achieve a goal. Instead of being given examples of desired behavior, the learning agent must discover by trial and error how to behave in order to get the most reward. The environment is a Markov decision process (MDP) with state set, $\mathcal{S}$, and action set, $\mathcal{A}$. The agent and the environment interact in a sequence of discrete steps, $t = 0, 1, 2, \ldots$ The state and action at one time step, $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$, determine the probability distribution for the state at the next time step, $s_{t+1} \in \mathcal{S}$, and, jointly, the distribution for the next reward, $r_{t+1} \in \Re$. The agent's objective is to chose each a_t to maximize the subsequent *return*:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k},$$

where the discount rate, $0 \leq \gamma \leq 1$, determines the relative weighting of immediate and delayed rewards. In some environments, the interaction consists of a sequence of episodes, each starting in a given state and ending upon arrival in a terminal state, terminating the series above. In other cases the interaction is continual, without interruption, and the sum may have an infinite number of terms (in which case we usually assume $\gamma < 1$). Infinite horizon cases with $\gamma = 1$ are also possible though less common (e.g., see Mahadevan, 1996).

The agent's action choices are a stochastic function of the state, called a *policy*, $\pi : \mathcal{S} \mapsto Pr(\mathcal{A})$. The *value* of a state given a policy is the expected return starting from that state following the policy:

$$V^\pi(s) = E\{R_t \mid s_t = s, \pi\},$$

and the best that can be done in a state is its optimal value:

$$V^*(s) = \max_\pi V^\pi(s).$$

There is always at least one *optimal policy*, π^*, that achieves this maximum at all states $s \in \mathcal{S}$. Paralleling the two *state*-value functions defined above are two *action*-value functions, $Q^\pi(s,a) = E\{R_t \mid s_t = s, a_t = a, \pi\}$ and $Q^*(s,a) = \max_\pi Q^\pi(s,a)$. From Q^* one can determine an optimal deterministic policy, $\pi^*(s) = \arg\max_a Q^*(s,a)$. For this reason, many RL algorithms focus on approximating Q^*. For example, *one-step tabular Q-learning* (Watkins, 1989) maintains a table of estimates $Q(s,a)$ for each pair of state and action. Whenever a is taken in s, $Q(s,a)$ is updated based on the resulting next state s', and reward r:

$$Q(s,a) \leftarrow (1-\alpha_{sa})Q(s,a) + \alpha_{sa}[r + \max_{a'} Q(s',a')], \qquad (1)$$

where $\alpha_{sa} > 0$ is a time-dependent step-size parameter. Under minimal technical conditions, Q converges asymptotically to Q^*, from which an optimal policy can be determined as described above (Watkins and Dayan, 1992).

Modern RL encompasses a wide range of problems and algorithms, of which the above is only the simplest case. For example, all the large applications of RL use not tables but parameterized function approximators such as neural networks (e.g., Tesauro, 1995; Crites and Barto, 1996; Singh and Bertsekas, 1997). It is also commonplace to consider planning—the computation of an optimal policy given a model of the environment—as well as learning (e.g., Moore and Atkeson, 1993; Singh, 1993). RL can also be used when the state is not completely observable (e.g., Loch and Singh, 1998). The methods that are effectively used in practice go far beyond what can be proven reliable or efficient. In this sense, the open theoretical questions in RL are legion. Here I highlight four that seem particularly important, pressing, or opportune. The first three are basic questions in RL that have remained open despite some attention by skilled mathematicians. Solving these is probably not just a simple matter of applying existing results; some new mathematics may be needed. The fourth open question concerns recent progress in extending the theory of uniform convergence and VC dimension to RL. For additional general background on RL, I recommend our recent textbook (Sutton and Barto, 1998).

1 Control with Function Approximation

An important subproblem within many RL algorithms is that of approximating Q^π or V^π for the policy π used to generate the training experience. This is called the *prediction* problem to distinguish it from the *control* problem of RL as a whole (finding Q^* or π^*). For the prediction problem, the use of generalizing function approximators such as neural networks is relatively well understood. In the strongest result in this area, the TD(λ) algorithm with linear function approximation has been proven asymptotically convergent to within a bounded expansion of the minimum possible error (Tsitsiklis and Van Roy, 1997). In contrast, the extension of Q-learning to linear function approximation has been shown to be unstable (divergent) in the prediction case (Baird, 1995). This pair of results has focused attention on Sarsa(λ), the extension of TD(λ) to form a control algorithm.

Empirically, linear Sarsa(λ) seems to perform well despite (in many cases) never converging in the conventional sense. The parameters of the linear function can be shown to have no fixed point in expected value. Yet neither do they diverge; they seem to "chatter" in the neighborhood of a good policy (Bertsekas and Tsitsiklis, 1996). This kind of solution can be completely satisfactory in practice, but can it be characterized theoretically? What can be assured about the quality of the chattering solution? New mathematical tools seem necessary. Linear Sarsa(λ) is thus both critical to the success of the RL enterprise and greatly in need of new learning theory.

2 Monte Carlo Control

An important dimension along which RL methods differ is their degree of *bootstrapping*. For example, one-step Q-learning bootstraps its estimate for $Q(s, a)$ upon its estimates for $Q(s', a')$ (see Eq. 1), that is, it builds its estimates upon themselves. Non-bootstrapping methods, also known as Monte Carlo methods, use only actual returns—no estimates—as their basis for updating other estimates. The λ in methods such as TD(λ), Q(λ), and Sarsa(λ) refers to this dimension, with $\lambda = 0$ (as in TD(0)) representing the most extreme form of bootstrapping, and $\lambda = 1$ representing no bootstrapping (Monte Carlo methods).

In most respects, the theory of Monte Carlo methods is better developed than that of bootstrapping methods. Without the self reference of bootstrapping, Monte Carlo methods are easier to analyze and closer to classical methods. In linear prediction, for example, Monte Carlo methods have the best asymptotic convergence guarantees. For the control case, however, results exist only for extreme bootstrapping methods, notably tabular Q(0) and tabular Sarsa(0). For any value of $\lambda > 0$ there are no convergence results for the control case. This lacunae is particularly glaring and galling for the simplest Monte Carlo algorithm, *Monte Carlo ES* (Sutton and Barto, 1998). This tabular method maintains $Q(s, a)$ as the average of all completed returns (we assume an episodic interaction) that started with taking action a in state s. Actions are selected greedily, $\pi(s) = \arg\max_a Q(s, a)$, while exploration is assured by assuming *exploring starts* (ES)—that is, that episodes start in randomly selected state–action pairs with all pairs having a positive probability of being selected. It is hard to imagine any RL method simpler or more likely to converge than this, yet there remain no proof of asymptotic convergence to Q^*. While this simplest case remains open we are unlikely to make progress on any control method for $\lambda > 0$.

3 Efficiency of Bootstrapping

Perhaps the single most important new idea in the field of RL is that of temporal-difference (TD) learning with bootstrapping. Bootstrapping TD methods have been shown empirically to learn substantially more efficiently than Monte Carlo methods. For example, Figure 1 presents a collection of empirical results in which λ was varied from 0 (pure bootstrapping) to 1 (no bootstrapping, Monte Carlo). In all cases, performance at 0 was better than performance at 1, and the best performance was at an intermediate value of λ. Similar results have been shown analytically (Singh and Dayan, 1998), but again only for particular tasks and initial settings. Thus, we have a range of results that suggest that bootstrapping TD methods are generally more efficient than Monte Carlo methods, but no definitive proof. While it remains unclear exactly what should or could be proved here, it is clear that this is a key open question at the heart of current and future RL.

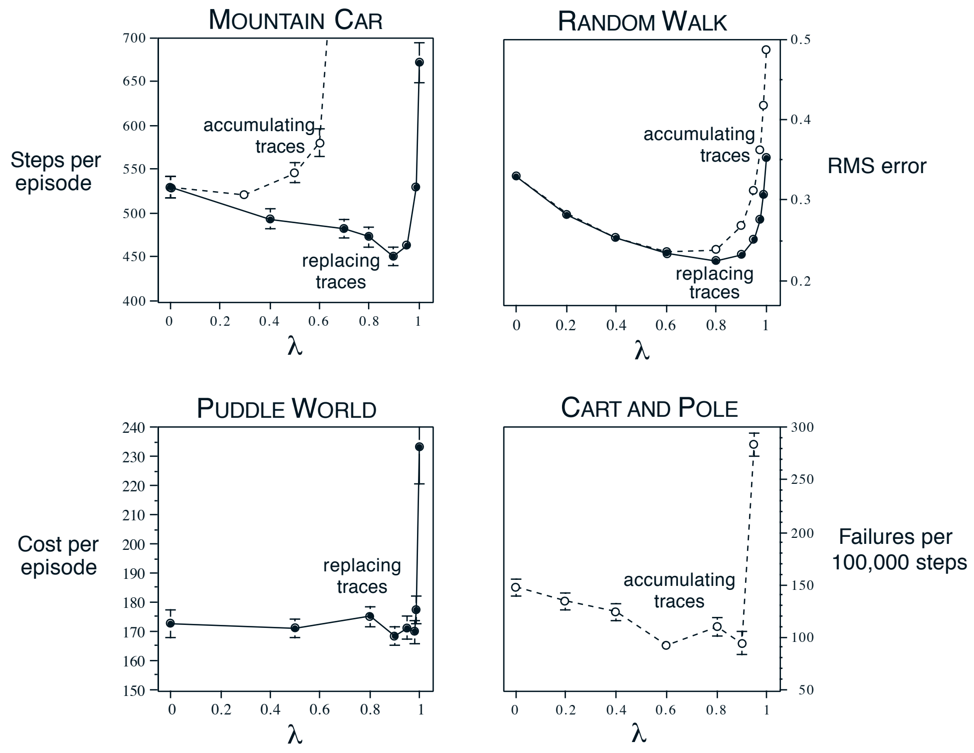

Fig. 1. The effect of λ on RL performance. In all cases, the better the performance, the *lower* the curve. The two left panels are applications to simple continuous-state control tasks using the Sarsa(λ) algorithm and tile coding, with either replacing or accumulating traces (Sutton, 1996). The upper-right panel is for policy evaluation on a random walk task using TD(λ) (Singh and Sutton, 1996). The lower right panel is unpublished data for a pole-balancing task from an earlier study (Sutton, 1984).

4 A VC Dimension for RL

So far we have discussed open theoretical questions at the heart of RL that are distant from those usually considered in computational learning theory (COLT). This should not be surprising; new problems are likely to call for new theory. But it is also worthwhile to try to apply existing theoretical ideas to new problems. Recently, some progress has been made in this direction by Kearns, Mansour and Ng (in prep.) that seems to open up a whole range of new possibilities for applying COLT ideas to RL.

Recall the classic COLT problem defined by a hypothesis space $\mathcal{H}$ of functions from $\mathcal{X}$ to $\mathcal{Y}$ together with a probability distribution $\mathcal{P}$ on $\mathcal{X} \times \mathcal{Y}$. Given a training set of x, y pairs chosen according to $\mathcal{P}$, the objective is to find a function $\hat{h} \in \mathcal{H}$ that minimizes the generalization error. A basic result establishes the number of examples (on the order of the VC dimension of $\mathcal{H}$) necessary to assure with high probability that the generalization error is approximately the same as the training error.

Kearns, Mansour and Ng consider a closely related planning problem in RL. Corresponding to the set of possible functions $\mathcal{H}$, they consider a set of possible policies Π. For example, Π could be all the greedy policies formed by approximating an action-value function with a neural network of a certain size. Corresponding to the probability distribution $\mathcal{P}$ on $\mathcal{X} \times \mathcal{Y}$, Kearns et al. use a generative or sample model of the MDP. Given any state s and action a, the model generates samples of the next state s' and the expected value of the next reward r, given s and a. They also allow the possibility that the environment is a partially observable (PO) MDP, in which case the model also generates a sample observation o, which alone is used by policies to select actions. Corresponding to the classical objective of finding an $\hat{h} \in \mathcal{H}$ that minimizes generalization error, they seek a policy $\hat{\pi} \in \Pi$ that maximizes performance on the (PO)MDP. Performance here is defined as the value, $V^{\hat{\pi}}(s_0)$, of some designated state state, s_0 (or, equivalently, on a designated distribution of starting states).

But what corresponds in the RL case to the training set of example x, y pairs? A key property of the conventional training set is that one such set can be *reused* to evaluate the accuracy of *any* hypothesis. But in the RL case different policies give rise to different action choices and thus to different parts of the state space being encountered. How can we construct a training set with a reuse property comparable to the supervised case? Kearns et al.'s answer is the *trajectory tree*, a tree of sample transitions starting at the start state and branching down along all possible action choices. For each action they obtain one sample next state and the expected reward from the generative model. They then recurse from these states, considering for each all possible actions and one sample outcome. They continue in this way for a sufficient depth, or horizon, H, such that γ^H is sufficiently small with respect to the target regret, ϵ. If there are two possible actions, then one such tree is of size 2^H, which is independent of the number of states in the (PO)MDP. The reuse property comes about because a single tree specifies a length H sample trajectory for any policy by working down the tree following the actions taken by that policy. A tree corresponds to a single example in the classic supervised problem, and a set of trees corresponds to s training set of examples.

With the trajectory tree construction, Kearns et al. are able to extend basic results of uniform convergence. The conventional definition of VC dimension cannot be directly applied to policy sets $\mathit{\Pi}$, but by going back to the original definitions they establish a natural extension of it. They prove that with (on order of) this number of trajectory trees, with probability δ, one can be assured of finding a policy whose value is within ϵ of the best policy in $\mathit{\Pi}$.

Kearns, Mansour and Ng's work breaks fertile new ground in the theory of RL, but it is far from finishing the story. Their work could be extended in many different directions just as uniform convergence theory for the supervised case has been elaborated. For example, one could establish the VC dimension on some policy classes of practical import, or extend boosting ideas to the RL case. Alternatively, one could propose replacements for the supervised training examples other than trajectory trees. Kearns et al. consider how trajectories from random policies can be used for this purpose, and there are doubtless other possibilities as well.

Acknowledgments

The author is grateful for substantial assistance in formulating these ideas from Satinder Singh, Michael Kearns, and Yoav Freund.

References

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth In-*

ternational Conference on Machine Learning, pp. 30–37. Morgan Kaufmann, San Francisco.

Bertsekas, D. P., and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA.

Crites, R. H., and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pp. 1017–1023. MIT Press, Cambridge, MA.

Kearns, M., Mansour, Y., Ng, A. Y. (in prep.). Sparse sampling methods for planning and learning in large and partially observable Markov decision processes.

Loch J., and Singh S. (1998). Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *Proceedings of the Fifteenth International Conference on Machine Learning.* Morgan Kaufmann, San Francisco.

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22:159–196.

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

Singh, S. P. (1993). *Learning to Solve Markovian Decision Processes.* Ph.D. thesis, University of Massachusetts, Amherst. Appeared as CMPSCI Technical Report 93-77.

Singh, S. P., and Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference*, pp. 974–980. MIT Press, Cambridge, MA.

Singh S., and Dayan P. (1998). Analytical mean squared error curves for temporal difference learning. *Machine Learning.*

Singh, S. P., and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* Ph.D. thesis, University of Massachusetts, Amherst.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pp. 1038–1044. MIT Press, Cambridge, MA.

Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA.

Tesauro, G. J. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38:58–68.

Tsitsiklis, J. N., and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge University.

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