Chattering in SARSA(λ)

A CMU Learning Lab Internal Report

Geoffrey J. Gordon
Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213
ggordon@cs.cmu.edu

April 24, 1996

1 INTRODUCTION

The SARSA(λ) algorithm attempts to solve a Markov decision process online. It does so by maintaining a linear approximation to the optimal Q function and updating it according to the TD(λ) rule along the trajectories it follows. Its chief distinction is that it only updates or examines a value Q_{xa} immediately after performing action a from state x. (Of course, its updates to Q_{xa} may generalize to other Q values elsewhere.) In order to ensure sufficient exploration, it must sometimes choose a non-greedy action; when it does so, the Q value for the greedy action is ignored and the non-greedy Q value is propagated backwards along the current trajectory.

In the current study, we assume that the agent explores by choosing a non-greedy action with some low probability ϵ, which either remains constant or is reduced slowly towards zero over the course of learning. We will also assume $\lambda = 0$, although it is possible to construct a similar example for any $\lambda \in [0, 1)$.

2 CHATTERING

The SARSA algorithm is not guaranteed to converge even in the benign case where the Q-function is approximated by state aggregation: when we apply SARSA(0) to the MDP in figure 1, one of the learned Q values oscillates forever. (We used a learning rate $\alpha = .01$ and no discount. To ensure sufficient exploration, the agent chose an apparently suboptimal action $\epsilon = 10\%$ of the time. Any other parameters would have resulted in similar behavior. In particular, discounting would change nothing; annealing α or ϵ to zero would merely cause the oscillations to get slower and slower while remaining at about the same amplitude.)

SARSA fails to converge for this MDP because the probability of visiting a given state can change discontinuously when the Q function fluctuates slightly.

When, by luck, the upper path through the MDP appears better, the cost-2 arc into the goal will be followed more often and the learned Q values will increase, while when the lower path appears better the cost-1 arc will be weighted more heavily and the Q values will decrease. In the cycle described below, each path appears better infinitely often and the learned Q values oscillate forever.

The phase plot in figure 1(c) shows a more detailed picture of what is happening in this example. Write Q_U and Q_L for the Q values of the upper and lower arcs out of the initial state; write Q_A for the aggregate Q value of the pair of arcs connected by a dotted line. Then the diagram plots $(Q_U - Q_L), Q_A$ as a function of number of trajectories experienced.

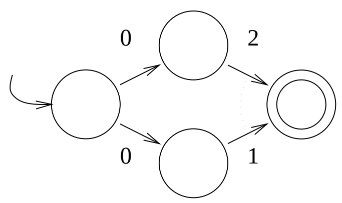
In what follows, we assume that both α and ϵ are very small, so that we can neglect second order effects. Reducing α and ϵ will increase the accuracy of this approximation.

We begin at the upper left corner, where all three Q values are approximately equal to 1.5 but $Q_U > Q_L > Q_A$. This state is unstable: with high probability, the agent will choose the lower path the next time it visits the initial state. When it does so, Q_L will tend to drop slightly, since its expected backed-up Q value is Q_A, which is slightly lower than Q_L. Next, Q_A will tend to drop strongly, since its current Q value of 1.5 is much higher than its expected backed-up value of 1. Now, when the agent returns to the initial state, it is again likely to choose Q_L; but this time, Q_A is farther below Q_L, so Q_L will tend to drop more strongly. This cycle will feed on itself: each time the agent chooses the lower path, Q_A will decay exponentially towards 1, pulling Q_L along with it. Since the agent chooses the lower path more often than the upper path, Q_L will have a higher effective learning rate than Q_U, so Q_L will tend to drop faster and the lower path will remain the greedy choice.

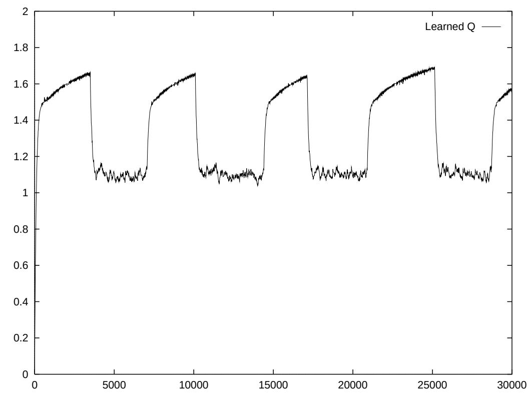
Eventually Q_A will decay to near 1, with Q_L not far behind. This state is represented by the lower right corner of the phase diagram, where $Q_A \approx 1$ and $Q_U - Q_L$ is positive. At this point, Q_U will slowly catch up with the other two Q values as the agent's exploration causes it to choose the upper path from time to time. During this time, Q_A will hover around $1+\epsilon$, its expected backed-up Q value when the upper and lower paths are followed with frequencies $\epsilon : (1-\epsilon)$. Q_L will remain approximately equal to Q_A, and Q_U will decay exponentially towards Q_A.

The resulting point in phase space is at the lower left of the diagram, where $Q_U - Q_L$ is near 0 and Q_A is near 1. The system will tend to hover here as long as $Q_U > Q_L$ so that the lower path is the greedy choice. The random exploration, however, will ensure that at some point Q_U will drop below Q_L.

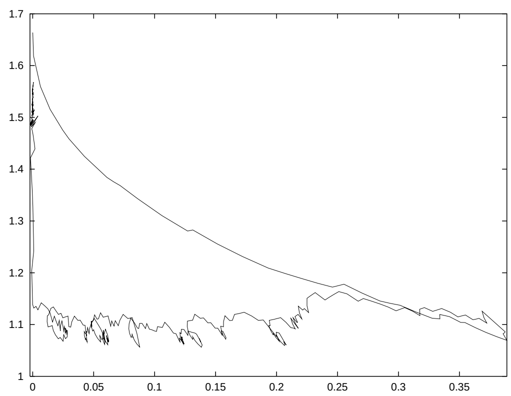
At this point, with high probability, the agent will choose the upper arc, and Q_A will jump upwards. On the next trajectory, Q_U will also tend to rise. Both Q_A and Q_U will continue to rise (with Q_A rising faster, since its target is farther away) until $Q_U > Q_L$. Once Q_U passes Q_L, the agent will tend to choose the lower path, so Q_L will tend to rise and Q_A will tend to fall. Crucially, as long as $Q_A < 1.5$, it will not fall as quickly as it had been rising (since the new target 1 is closer than the old target 2). Therefore, by the time Q_L passes Q_U again, Q_A will with high probability still be above both Q_L and Q_U.



(a)



(b)



(c)

Figure 1: (a) An absorbing MDP: from the start state, the agent may choose the upper or the lower path, but from then on its decisions are forced. Next to each arc is its cost. The arcs connected by a dotted line are aggregated, so that the agent must learn the same Q value for each one. (b) The learned Q value for the aggregated arcs during the first 30 000 trajectories. (c) A phase plot of a single oscillation. On the Y axis is the learned Q value for the aggregated arcs, just as in (b); on the X axis is the difference between the other two learned Q values.

In this manner, the agent will alternate between the upper and lower paths, with Q_L always approximately equal to Q_U and $Q_L, Q_U < Q_A$. As long as Q_A is significantly below 1.5, the agent will follow the upper and lower paths nearly equally frequently: since Q_L and Q_U are both moving towards Q_A, it will take about as many updates for Q_L to pass Q_U as it will for Q_U to pass Q_L.

Finally, we will again reach the upper left corner of the phase diagram: as Q_A rises more and more slowly, Q_L and Q_U will eventually catch up with it. Once they do, a single traversal of the lower path can move Q_A below Q_L and Q_U, starting the first phase of the cycle over again.

The above cycle remains substantially the same for any sufficiently small values of α and ϵ. In particular, the distance from each corner of the cycle to the next approaches a strictly positive value as $\alpha, \epsilon \to 0$. Smaller values of α cause the number of iterations required for a cycle to increase; smaller values of ϵ cause the second phase of the cycle (moving from lower right to lower left) to require more iterations, but do not substantially affect the time required for the other two phases of the cycle.

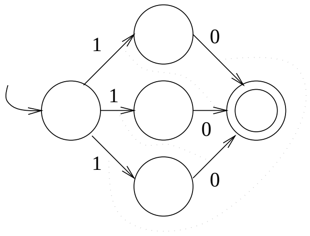
In general, we will choose a different learning and exploration rate for each trajectory, and decay both α and ϵ towards 0 over the course of learning (subject to the requirement that $\sum_t |\alpha_t| = \infty$ and $\sum_t |\epsilon_t| = \infty$, to ensure sufficient learning and exploration). In this case, the learned Q function will continue to cycle through its trajectory more and more slowly; but at each point the steady-state variance of Q_U will be bounded below by some constant, so the Q function will not converge.

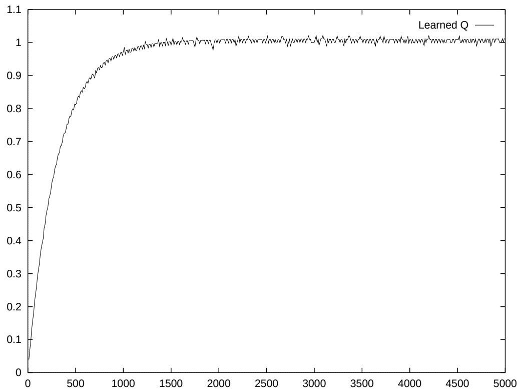
3 HOW GENERAL IS THIS PROBLEM

The problem described above happens because SARSA's policy is a discontinuous function of the current learned Q values. We can divide the space of all Q functions into regions where SARSA follows a constant policy. These regions will be convex since each one is the solution to a set of inequalities of the form $Q_{xa} \leq Q_{xb}$, where a is the greedy action from state x. (Since each Q value is represented as a linear combination of weights, these regions will also be convex in weight space.)

Bertsekas has called these sets "greedy regions" because each one corresponds to a different greedy policy. Each greedy region has its own "greedy point," the Q function to which TD(λ) will converge if the agent continues to follow the region's policy. If a greedy region strictly contains its own greedy point, then that point is stable for SARSA: for any positive δ, there will be a neighborhood from which SARSA will converge with probability at least $(1-\delta)$ to the stable point, as long as α and ϵ start out small enough and are annealed appropriately to 0.

The MDP of the previous section has exactly two greedy regions and two greedy points. SARSA oscillates because the greedy points lie on the boundary between the greedy regions. This example is just a special case of a more general phenomenon: it is possible to construct a pair of greedy regions, each





(a) (b)

Figure 2: (a) A Markov decision process. The learning rate, discount factor, and exploration probability are as before. The pairs of arcs connected by dotted lines are constrained to learn identical Q values. (b) The learned Q value for one of the arcs over the first 5 000 trajectories.

of which strictly contains the other's greedy point. (In fact it is possible to construct a cycle of many greedy regions, each containing the previous one's greedy point—see figure 2.) In such a case SARSA may converge to a point on the border between the greedy regions, as it does in the MDP of figure 2, or it may cycle between these (and possibly other) greedy regions. Bertsekas has reported an example of convergence to a point on the border between greedy regions; his example is based on modified policy iteration rather than SARSA, but the principle is the same.

If our function approximator is powerful enough to represent every Q function exactly, examples such as those in figures 1 and 2 are no longer possible. With a complete function approximator, each greedy region in a cycle must have the same greedy point (in fact, this point will be the optimal Q function, and it will be on the boundary of every greedy region in the cycle). It is an open question whether SARSA or any similar algorithm converges to the optimal policy when the function approximator is complete.

4 Sources

Temporal difference learning was introduced in [Sut88]. The idea of a Q function was presented in [Wat89]. Various authors including Sutton, Dayan, and Sejnowski have proven the convergence of TD(λ) in various situations; the most comprehensive proof is [TV96]. A more complete description of SARSA, together with examples, is in [Sut96]. The example of figure 1 is a simplified version of an example presented in [Gor96]. The examples attributed to Bertsekas were presented in his talk at the NSF workshop on reinforcement learning [Ber96].

Acknowledgements

This material is based on work supported under a National Science Foundation Graduate Research Fellowship and by ARPA grant number F33615-93-1-1330. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation, ARPA, or the United States government.

References

[Ber96] D. Bertsekas. Talk, 1996. Given at the NSF workshop on reinforcement learning.

[Gor96] G. J. Gordon. Stable fitted reinforcement learning. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1996.

[Sut88] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[Sut96] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1996.

[TV96] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. Technical Report LIDS-P-2322, MIT, 1996.

[Wat89] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.