Online Fitted Reinforcement Learning

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

Abstract

My paper in the main portion of the conference deals with fitted value iteration or Q-learning for offline problems, *i.e.*, those where we have a model of the environment so that we can examine arbitrary transitions in arbitrary order. The same techniques also allow us to do Q-learning for an online problem, *i.e.*, one where we have no model but must instead perform experiments inside the MDP to gather data. I will describe how.

1 INTRODUCTION

For a more detailed coverage of the material in this introduction, see (Gordon, 1995a, Gordon, 1995b).

Suppose we have a discounted Markov decision process M. Let T_M be the Q-learning backup operator for M. Let A be an averaging function approximator with associated mapping M_A. Then the fitted backup operator $M_A \circ T_M$ is a contraction mapping, so that iteration of this operator always converges.

The fitted backup operator $M_A \circ T_M$ is useful for offline dynamic programming only if it is much cheaper to compute than the exact backup operator T_M. One way to make sure that the fitted operator is cheap is to require that M_A pay attention only to a small subset of all possible (state, action) pairs: in this case, there is no need to compute the other components of T_M, because they will be ignored anyway.

For online reinforcement learning, a function approximator A which ignores most (state, action) pairs is inappropriate: since we no longer have complete control over which transitions we observe, and since such an A derives information only from a small fraction of the possible transitions, we will be forced to ignore most of our training data. So, for online learning, we want M_A to depend on every (state, action) pair rather than just a few, so that we can derive information from every transition that we observe.

The question, then, is how to keep track of the learned Q function efficiently. For an offline problem, we simply remembered the Q values for the (state, action) pairs that M_A payed attention to. We can no longer do so when M_A pays attention to all states and actions. The next section describes a way to keep track of the Q function for online learning.

2 FEATURE-BASED APPROXIMATORS

In this section, we will present a simple randomized algorithm for offline learning, then modify it until it is appropriate for online problems as well. The final algorithm will collect data by following a fixed exploration policy π, which is assumed to be given; but it will compute an approximation to Q^*, the optimal Q function, rather than to Q^π. Just as in (Gordon, 1995a), the algorithm can be viewed as approximating the solution to our original MDP by solving a simpler MDP exactly; but now, since we don't know the original MDP, we will be unable to compute the simpler MDP.

The algorithm of this section is, to our knowledge, the first online reinforcement learning algorithm for general MDPs which can use function approximators stronger than state aggregation while maintaining a convergence guarantee.

Suppose that we divide the approximation mapping M_A into two stages, an encode and a decode, so that $M_A = M_A^d \circ M_A^e$. The encode stage, M_A^e, will estimate the values of some number of features from the target Q function, while the decode stage, M_A^d, will reconstruct the fitted Q function from the values of the features. For offline problems, M_A^e will just pick out the values of a few selected (state, action) pairs, while for online problems M_A^e will build its features by averaging many such pairs. For any given M_A, there may be many such decompositions; we will assume that the chosen decomposition is such that M_A^e and M_A^d are monotone linear max-norm nonexpansions.

Any mapping M_A can be so decomposed: at the very least, we can take M_A^e to be the identity (so that the value of each (state, action) pair is a separate feature) and $M_A^d = M_A$. In this paper, though, we are particularly interested in those mappings which can be decomposed using only a few features. That is, if M has n states and m actions, and if M_A uses k features, then $M_A^e \in (\mathbb{R}^{nm} \mapsto \mathbb{R}^k)$ and $M_A^d \in (\mathbb{R}^k \mapsto \mathbb{R}^{nm})$ for $k \ll nm$.

Let x be a state, a be an action, and f be a feature. Let q be a vector in $\mathbb{R}^{nm}$ and θ a vector in $\mathbb{R}^k$. Then we will define p^e_{fxa} and p^d_{xaf} as follows:

$$\begin{aligned}[M_A^e(q)]_f &= \sum_x \sum_a p^e_{fxa} q_{xa} \\ [M_A^d(\theta)]_{xa} &= \sum_f p^d_{xaf} \theta_f\end{aligned}$$

In the notation of (Gordon, 1995b), these definitions mean that the coefficients of M_A are

$$\beta_{xayb} = \sum_f p^d_{xaf} p^e_{fyb}$$

(Note that we have omitted the constants β_{xa} and k_{xa} from our notation, so that $\sum_y \sum_b \beta_{xayb} = 1$. This omission involves no loss of generality, since we can always take care of these constants by adding dummy states whose values are fixed appropriately.) The assumption that M_A^e and M_A^d are max-norm nonexpansions now translates to the equations $\sum_x \sum_a p^e_{fxa} = 1$ and $\sum_f p^d_{xaf} = 1$.

Consider the following algorithm: for t from 1 onward, choose a feature f_t uniformly, then generate a state x_t and action a_t with probability $p^e_{f_t x_t a_t}$. Next, observe a transition from state x_t under action a_t with cost c_t to state y_t. Finally, if θ is the current vector of feature values, perform the Q-learning-like update

$$\theta_f \leftarrow_{\alpha_{tf}} c_t + \gamma \min_b \sum_g p^d_{y_t b g} \theta_g$$

Here α_{tf} is a learning rate, which is assumed to be defined for all f at each t, but to be nonzero for only f_t at time t. The expected value of this update is

$$\begin{aligned}&E(c_t + \gamma \min_b \sum_g p^d_{y_t b g} \theta_g) \\ &= \sum_x \sum_a p^e_{fxa} (c_{xa} + \gamma \sum_y p_{xay} \min_b \sum_g p^d_{ybg} \theta_g)\end{aligned}$$

If we compute this update for all features f and write the result in matrix form, we have that the expected update to θ is

$$M_A^e(T_M(M_A^d(\theta)))$$

Since T_M is a max-norm contraction with factor γ and M_A^e and M_A^d are nonexpansions, the compound operator $M_A^e \circ T_M \circ M_A^d$ is a max-norm contraction with factor γ; therefore this operator has a unique fixed point $\tilde{\theta}$, and theorem 1 of (Jaakkola *et al.*, 1994) shows that the above algorithm converges with probability 1 to $\tilde{\theta}$ as long as for all f the learning rates satisfy

$$\sum_t \alpha_{tf} = \infty \qquad \sum_t \alpha^2_{tf} < \infty$$

Note that $M_A^d(\tilde{\theta})$ is the fixed point of $M_A \circ T_M$, so the analyses of (Gordon, 1995a) apply.

We can modify the above algorithm slightly as follows: suppose that, on each time step t, we first generate a state x_t and action a_t with probability $p_{x_t a_t}$, then generate a feature f_t with probability $p'_{f_t x_t a_t}$. Then Bayes' rule gives us the probability of seeing a particular state and action on time step t, given the value of f_t:

$$P(x_t = x, a_t = a \mid f_t = f) = \frac{p'_{fxa} p_{xa}}{\sum_y \sum_b p'_{fyb} p_{yb}} \equiv p^{e'}_{fxa}$$

This modified algorithm is essentially the same as the original one; the main difference is that it would be time-consuming to compute the probabilities $p^{e'}_{fxa}$. In other words, we have substituted some effectively unknown encode stage $M_A^{e'}$ into our function approximator while leaving the decode stage the same. (Actually, $M_A^{e'}$ is not completely unknown: for example, $p^{e'}_{fxa}$ can only be nonzero if p'_{fxa} is. Also, one might hope that, if the states and actions being averaged together to find the value of feature f really are similar, the exact coefficients of the average might not matter too much.)

Of course, we cannot apply either of the above algorithms to an online problem, because they require observing a transition from an arbitrary state at each time step. We can, however, do the following (see (Singh *et al.*, 1995) for a similar analysis). Suppose that we follow a fixed exploration policy π in M. Then there will be a fixed limiting frequency p^π_{xa} with which we will visit each state and action. Just as before, we can generate a feature randomly on each step with probabilities p'_{fxa} and perform the above Q-learning-like backup. We will let $p^{e,\pi}_{fxa}$ be the limiting frequency of visits to state x and action a given that the current feature is f, and we will let $M_A^{e,\pi}$ be the mapping with coefficients $p^{e,\pi}_{fxa}$. (Note that these frequencies are impossible to compute without knowledge of M.) If the states and actions on each step were independent, then we could again apply theorem 1 from (Jaakkola *et al.*, 1994) to show convergence.

The observations on different time steps are not in fact independent; but for any ϵ we can choose a sufficiently large M that, over an M-step interval, the expected frequency of visits to any state x and action a while updating feature f will be within ϵ of $p^{e,\pi}_{fxa}$. That means that, if we collect updates for M steps before applying them, the expected update to θ_f is

$$\sum_x \sum_a (p^{e,\pi}_{fxa} + \epsilon_{fxa}) [T_M(M_A^d(\theta))]_{xa}$$

for some constants $-\epsilon \leq \epsilon_{fxa} \leq \epsilon$.

We can compute the following bound for the difference between $\tilde{\theta}_f$ and the expected update to θ_f:

$$\begin{aligned}
&\left|\left(\sum_x \sum_a (p^{\epsilon,\pi}_{fxa} + \epsilon_{fxa})[T_M(M^d_A(\theta))]_{xa}\right) - \tilde{\theta}_f\right| \\
&\leq \left|\sum_x \sum_a \epsilon_{fxa}[T_M(M^d_A(\theta))]_{xa}\right| + \\
&\quad \left|\left[M^{\epsilon,\pi}_A(T_M(M^d_A(\theta)))\right]_f - \tilde{\theta}_f\right| \\
&\leq \left|\sum_x \sum_a \epsilon_{fxa}\left[T_M(M^d_A(\theta)) - T_M(M^d_A(\tilde{\theta}))\right]_{xa}\right| + \\
&\quad \left|\sum_x \sum_a \epsilon_{fxa}\left[T_M(M^d_A(\tilde{\theta}))\right]_{xa}\right| + \gamma\|\theta - \tilde{\theta}\| \\
&\leq C\epsilon\left|\left[T_M(M^d_A(\theta)) - T_M(M^d_A(\tilde{\theta}))\right]_{xa}\right| + \\
&\quad C\epsilon\left\|T_M(M^d_A(\tilde{\theta}))\right\| + \gamma\|\theta - \tilde{\theta}\| \\
&\leq (1 + C\epsilon)\gamma\|\theta - \tilde{\theta}\| + C\epsilon\gamma\|\tilde{\theta}\|
\end{aligned}$$

for some constant C.

For sufficiently small ϵ, this inequality means that there exists a $\gamma' < 1$ and an ϵ' so that the difference between $\tilde{\theta}_f$ and the expected update to θ_f is less than $(\epsilon' + \gamma'\|\theta - \tilde{\theta}\|)$. A minor modification to theorem 1 of (Jaakkola *et al.*, 1994) now shows that (for appropriate learning rates) the algorithm converges with probability 1 to the region

$$\left\{\theta \;\middle|\; \|\theta - \tilde{\theta}\| \leq \frac{\epsilon'}{1 - \gamma'}\right\}$$

By increasing M, we can make $\gamma' \to \gamma$ and $\epsilon' \to 0$, so that this region becomes as small as necessary.

Finally, by an analysis similar to the one in theorem 3 of (Jaakkola *et al.*, 1994), we can show that the difference between updating every step and accumulating the updates for M steps vanishes in the limit. The reason for this effect is that, for any starting point θ, once the learning rate is small enough, the difference between the online update to θ and the M-step update to θ is dwarfed by the difference between θ and $M^{\epsilon,\pi}_A(T_M(M^d_A(\theta)))$. The fact that online updating is equivalent in the limit to M-step updating for all M means, in turn, that M-step and N-step updating are equivalent; that is, no matter how frequently or infrequently we perform the updating, we always reach $\tilde{\theta}$ eventually.

An interesting early version of this algorithm was described in (Gordon, 1995b). According to that algorithm, the agent would begin with an offline approximator such as k-nearest-neighbor that pays attention only to some sample of states X_0. After observing a transition out of some state $x \notin X_0$, the agent would find the nearest state $x' \in X_0$, pretend that the transition actually originated from x', and make a Q-learning backup accordingly. The paper presented no convergence analysis, and in fact, since the algorithm did not follow a fixed exploration policy, it could oscillate forever.

References

G. J. Gordon. Stable function approximation in dynamic programming. In *Machine Learning (proceedings of the twelfth international conference)*, San Francisco, CA, 1995. Morgan Kaufmann.

G. J. Gordon. Stable function approximation in dynamic programming. Technical Report CS-95-103, CMU, 1995.

T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 7. Morgan Kaufmann, 1995.