Notes on Dynamic Programming and Reinforcement Learning

Jacob W. Crandall

November 19, 2012

1 Introduction

These notes contain a brief introduction to dynamic programming and reinforcement learning. The notes are not meant to be all-inclusive, but simply a brief introduction. For a more thorough investigation of single-agent reinforcement learning, I suggest that your read Sutton and Barto's book [2] and/or Kaelbling's survey [1]. Both can be found online.

We note that these methods were created for stationary (single-agent) environments. In general, the optimality guarantees of these algorithms do not apply to environments affected by agents that are also adapting to the environment. However, that has not kept researchers from applying these methods (and sometimes successfully) in multi-agent environments. We will talk a bit about that in a subsequent lecture.

2 Dynamic Programming

Dynamic programming was initially proposed by Bellman back in the 1940s and 50s. The term does not really make sense, but it sounds cool anyway. Essentially, dynamic programming refers to a method for finding the optimal control by breaking the problem into a slightly smaller subproblem (or subproblems).

Dynamic programming is well-suited for situations that can be modeled as Markov Decision Processes (MDPs). A MDP consists of a set of states S which describe the various situations in the world that an agent can be in, a set of actions A that the agent can carry out in each state, a transition function $\mathcal{T} : S \times A \rightarrow S$, which determines the new state of the world $s' \in S$ after action $a \in A$ is taken in state $s \in S$ (this transition function can be deterministic or probabilistic), and a reward function $\mathcal{R} : S \times A \rightarrow R$, which specifies the immediate reward r_t that the agent receives for taking action $a \in A$ in state $s \in S$.

In words, an agent that is state $s \in S$, must select an action $a \in A$. The action chosen and taken by the agent has two consequences. First, the agent receives some immediate reward r_t for taking that action. Second, the world transitions to a new state s', from which the agent must again select an action.

An agent acting in an MDP tries to maximize its future discounted reward. Remember that a strategy is a specification of what the agent will do in each state of the world. Thus, the goal of the agent is to select strategy π such that the total future discounted reward $R = \sum_t \gamma^{t-1} r_t$ is maximized:

$$\pi^* = \arg\max_{\pi} \sum_t \gamma^{t-1} r_t \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor, and the summation is over all time periods. Recall that each r_t depends on the state of the world at time t (denoted s_t) and the action the agent selects (using its strategy π) at time t (denoted a_t).

To determine a strategy that maximizes its sum of future discounted rewards, all an agent needs to do is determine the expected utility (future discounted reward) of taking each action $a \in A$ in each state $s \in S$, and then choose the action with the highest expected utility in each state. Let $Q(s, a)$ denote the utility of taking action a in state s given that the agent plays *optimally* (with respect to its future discounted payoffs) in subsequent states. We must determine how to compute $Q(s, a)$.

When we consider future discounted rewards, $Q(s, a)$ is given by the following equation:

$$Q(s, a) = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3, \ldots, \tag{2}$$

where $r_0 = r(s, a)$ is the immediate reward to the agent after taking action a in state s, r_1 is the immediate reward received in the next time step, etc. We can simplify this equation by lumping the discounted sum of future rewards r_1 through r_∞ into a single value $V(s')$, which is the value of being in the subsequent state s'. Then,

$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in S} \mathcal{T}(s, a, s')V(s'), \tag{3}$$

where $\mathcal{T}(s, a, s')$ is the probability of the state of the world becoming s' after action a is taken in state s. Eq. (3) is a form of the famous Bellman equation.

3 Value Iteration

Eq. (3) says that I can determine how much future discounted reward I will get if I take action a from state s if I know the reward function $r(s, a)$, the value function $V(s')$ for each s', and the transition function $\mathcal{T}(s, a, s')$. If I know what $Q(s, a)$ is for each $a \in A$, I can easily pick which action will give me the highest reward. That is, my optimal strategy in each state is given by $\pi^* = \arg\max_{a \in A} Q(s, a)$.

If I know the reward function $r(s, a)$ and the transition function $\mathcal{T}(s, a, s')$, then I can easily compute $Q(s, a)$ using *value iteration*. The value iteration algorithm is given in Algorithm 1.

Algorithm 1 Value iteration – assumes knowledge of $r(s, a)$ and $\mathcal{T}(s, a, s')$

For all $s \in S$, initialize $V(s)$ arbitarily
repeat
  Loop through each $s \in S$
    Loop through each $a \in A$
      $Q(s, a) = r(s, a) + \gamma \sum_{s' \in S} \mathcal{T}(s, a, s')V(s')$
    $V(s) = \max_{a \in A} Q(s, a)$
until convergence criteria met

The convergence criteria of the algorithm can be lots of things, but often we might say that the system has converged when the sum of changes over all Q-values is less then some threshold.

4 Model-Based Reinforcement Learning

But what if we don't know $r(s, a)$ and $\mathcal{T}(s, a, s')$? In that case, we can estimate $r(s, a)$ and $\mathcal{T}(s, a, s')$ through experience, and then compute the Q-values using value iteration and those estimates. To estimate the $r(s, a)$ and $\mathcal{T}(s, a, s')$, we simply act in the world and then record what happens. We can estimate $\mathcal{T}(s, a, s')$ by simply counting how many times I reach state s' when I take action a in state s. Similarly, we can estimate $r(s, a)$ as the mean reward received immediately after taking action a in state s. Each time we update $r(s, a)$ and $\mathcal{T}(s, a)$ (or every few times), we can update our Q-estimates using value iteration.

One trick is to adequately sample the environment so that we adequately estimate the functions, but eventually start using the knowledge we have. One can envision two potential problems. First, we could explore (act randomly, for example) the environment so long that we never exploit what we learn. Alternately, we could exploit our early estimates of $r(s, a)$ and $\mathcal{T}(s, a, s')$ so much that we never accurately model the environment, which could cause us to converge to a strategy that is suboptimal.

The most basic way of balancing these two issues is through what is known as ε-greedy exploration. With probably $1 - \varepsilon$, where $\varepsilon \in [0, 1]$, we follow our current estimate of π^*. With probability ε, we act randomly. Over time, we typically decrease the exploration rate ε. Formally, the agent's strategy is:

$$\pi_t(s) = \begin{cases} \arg\max_{a \in A} Q(s, a) & \text{with probability } 1 - \varepsilon \\ \text{random} & \text{otherwise} \end{cases} \tag{4}$$

5 Model-Free Reinforcement Learning

Another way of estimating the Q-values is using model-free reinforcement learning. The most popular model-free reinforcement learning algorithm is Q-learning. In Q-learning, an agent starts with some (usually random) estimation of $Q(s, a)$, which we denote $Q_0(s, a)$, for each action a in each state s. Each time the agent takes action a in state s, it observes the immediate reward $r_t = r(s, a)$ that follows, as well as the subsequent state s'. It then updates $Q_0(s, a)$ using the following Q-update:

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha_t \left(r(s, a) + \gamma V_t(s') - Q_t(s, a) \right), \tag{5}$$

where $\alpha_t \in [0, 1]$ is the agent's learning rate at time t, and $V_t(s')$ is given by:

$$V_t(s') = \max_{a \in A} Q_t(s, a). \tag{6}$$

In words, each time the agent takes action a from state s, it updates the estimate of its future discounted reward for taking action a in state s by moving its estimate incrementally toward the estimated discounted future reward it receives in this particular instance (i.e., $r(s, a) + \gamma \max_{a \in A} Q_t(s, a)$).

ε-greedy exploration is also used in Q-learning to converge to optimal strategies. In MDPs in which the environment is stationary, Q-learning is guaranteed to converge to "true Q-values" if two conditions hold [3]:

1. The learning rate α_t must decrease quickly, but not too quickly (I'm going to leave this rather vague), and
2. Each state-action pair (s, a) is visited infinitely often.

Practically speaking, these conditions (mostly the second) are impractical. Furthermore, model-free reinforcement learning is very slow, so it often cannot be used effectively in online learning problems.

6 Multi-agent Reinforcement Learning

As you have probably noted, multi-agent environments are not stationary in general. However, that has not kept researchers from trying to make them work in such environments. In the next class, we'll talk about their efforts.

References

[1] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–277, 1996.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[3] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.