CS 181: Markov Decision Processes and Reinforcement Learning

Carl Denton, Ankit Gupta, Jeffrey Ling

Week of April 17, 2017

1 Markov Decision Processes

A Markov Decision Process (MDP) is a framework for modeling an agent's actions in the world. It consists of:

1. A set of states S
2. A set of actions A
3. A reward function $r : S \times A \to \mathbb{R}$
4. A transition model $p(s'|s, a)$, $\forall s, s' \in S, a \in A$.

A *policy* π is a mapping from states to actions, i.e. $\pi : S \to A$.

1.1 Finite time horizon MDP

In the finite horizon setting, a policy may vary with the number of time periods remaining. $\pi_{(t)}$ denotes the policy with t time steps to go. T is the decision horizon. The value of a policy with t time steps to go is defined inductively to be:

$$V^{\pi}_{(t)}(s) = \begin{cases} r(s, \pi_{(1)}(s)) & \text{if } t = 1 \\ r(s, \pi_{(t)}(s)) + \sum_{s' \in S} p(s'|s, \pi_{(t)}(s)) V^{\pi}_{(t-1)}(s') & \text{o.w.} \end{cases} \tag{1}$$

The process of computing these values inductively, working from the end of the horizon to the present, is called *value iteration*. If we instead look forward in time, we are computing the expected value of the policy

$$V^{\pi}(s) = \mathbb{E}_{s_1,\ldots,s_T}\left[\sum_{t=1}^{T} r(s_t, \pi_{(T+1-t)}(s_t))\right] \tag{2}$$

by induction, where $s_1 := s$. $V^{\pi}(s)$ is the MDP value function.

In an MDP, the general goal is to find an optimal policy by maximizing the expected reward under the policy, i.e. maximizing the value function. This is the planning problem.

1.2 Infinite Horizon MDP

Policy Evaluation We can also send $T \to \infty$, i.e. have an infinite time horizon. In that case, we need a discount factor $0 < \gamma < 1$, and we want to compute the value function

$$V^{\pi}(s) = \mathbb{E}_{s_1, s_2, \ldots}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, \pi(s_t))\right] \tag{3}$$

where $s_1 := s$, and the γ factor ensures convergence (assuming bounded rewards). In this setting, we only worry about stationary policies that don't vary with time. This is the policy evaluation problem; for any given policy π, we can find $V^{\pi}(s)$ by solving the system of linear equations

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^{\pi}(s') \tag{4}$$

These capture consistency about the value function. To solve this system, we can use Gaussian elimination, or simply iterate until convergence as in the finite horizon case.

Value Iteration Suppose we have an optimal policy π^*. This satisfies the following set of equations known as the Bellman equations:

$$V^*(s) = \max_{a \in A}\left[r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^*(s')\right] \tag{5}$$

where $V^* \triangleq V^{\pi^*}$. Assuming we know V^*, we can read off the optimal policy by setting

$$\pi^*(s) = \arg\max_{a \in A}\left[r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^*(s')\right] \tag{6}$$

In order to find V^*, we can use value iteration:

- Initialize: $V(s) = 0$ for all states s.
- Update step (Bellman operator):

$$V'(s) = \max_{a \in A}\left[r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V(s')\right], \ \forall s \tag{7}$$

- $V \leftarrow V'$

where we iterate until convergence of V, which is guaranteed. With our converged V, we can then find π^* as in Equation 6.

Policy Iteration Another approach to planning is called *policy iteration*. To do policy iteration, we evaluate a proposed policy π by finding V^π as in Equation 4. This is Evaluation step (E step). Then, we do a policy improvement step (I step) by the equation

$$\pi'(s) \leftarrow \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^\pi(s') \right], \quad \forall s \tag{8}$$

We repeat the E and I steps until the policy π converges (stops changing).

Note that policy iteration takes more computation per iteration, but tends to converge faster in practice.

2 Reinforcement Learning

In the general reinforcement learning setting, we don't have access to the transition distribution $p(s'|s,a)$ or the reward function $r(s,a)$ — information about these only come to us through the outcome of the environment. This problem is hard because some states can lead to high rewards, but we don't know which ones; even if we did, we don't know how to get there!

To deal with this, in lecture, we discussed *model-based* and *model-free* reinforcement learning.

2.1 Model-based Learning

For model-based learning, we estimate the missing world models: $r(s,a)$ and $p(s'|s,a)$, and then utilize planning (value or policy iteration) to develop a policy π.

Exercise 1: Model Building

Question: How would we do this in practice? What is the downside of this approach?

2.2 Model-Free Learning

In model-free learning, we are no longer interested in learning the transition function and reward function. Instead, we are looking to directly infer the optimal policy from samples of the world — that is, given that we are in state s, we want to know the best action $a = \pi^*(s)$ to take.

To do this, we use an alternate formulation of the Bellman equations, which states that for an optimal policy π^*,

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} [Q^*(s',a')], \ \forall s, a \tag{9}$$

Then, once we have Q^*, we note that

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{10}$$

The question then becomes how we can find the Q values that satisfy the Bellman equations as written in Equation 9. To do this, we parameterize $Q(s,a;\mathbf{w})$ with parameters $\mathbf{w}$, where $w_{s,a} = Q(s,a;\mathbf{w})$ is a table of estimated Q values. We define a loss function which we want to minimize:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\mathbb{E}_{s,a}\left(Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q(s',a';\mathbf{w})]\right)^2 \tag{11}$$

We can optimize this via stochastic gradient descent. In order to get samples to perform gradient descent on, the idea is to approximate the loss by sampling s, a, giving a single gradient descent step

$$\frac{\partial \mathcal{L}}{\partial w_{s,a}} = Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q(s',a';\mathbf{w})] \tag{12}$$

If we can obtain a sample for the next state s' from the environment, we can approximate this as

$$\frac{\partial \mathcal{L}}{\partial w_{s,a}} \approx Q(s,a;\mathbf{w}) - [r + \gamma \max_{a'} Q(s',a';\mathbf{w})] \tag{13}$$

This is the term that we will use to update our estimates for Q:

$$w_{s,a} \leftarrow w_{s,a} - \eta \frac{\partial \mathcal{L}}{\partial w_{s,a}} \tag{14}$$

where $\eta > 0$ is the learning rate.

2.3 Exploration vs. Exploitation

How do we actually obtain the above samples to approximate the loss? The key issue here is exploration vs. exploitation.

In an exploitative approach, when we are in state s, we can simply take action $a = \arg\max_{a \in A} Q(s, a; \mathbf{w})$ based on our current estimate of the Q-function. However, if our estimate of the Q-function is bad, we might never find the true best action this way.

In an explorative approach, we might pick an action at random — one method is ϵ-greedy, where we take $\arg\max_{a \in A} Q(s, a; \mathbf{w})$ with probability $1-\epsilon$, and pick $a \in A$ uniformly at random with probability ϵ (for some $\epsilon > 0$).

We want to balance these two — exploitation lets us arrive at the optimal policy faster, but exploration is necessary to properly understand the state space. In practice, ϵ-greedy is a good policy for minimizing $\mathcal{L}$.

2.4 On-Policy vs. Off-Policy

Suppose we now have a policy π that balances exploration and exploitation (e.g. ϵ-greedy). To satisfy the Bellman equations for $Q^*(s, a)$, the policy would need to take the optimal actions in all future steps. Obviously, this won't always be true if we are exploring.

Therefore, there are multiple ways to write the gradient estimate $\partial \mathcal{L} / \partial w_{s,a}$. These are about how to learn, not about which explore/exploit policy to follow. One approach is on-policy:

$$\frac{\partial \mathcal{L}}{\partial w_{s,a}} \approx Q(s, a; \mathbf{w}) - [r + \gamma Q(s', \pi(s'); \mathbf{w})] \tag{15}$$

This is known as the SARSA update (State-Action-Reward-State-Action), since we look ahead to get the action $\pi(s')$. Since we follow π in choosing this action, this gradient method is on-policy. In practice this converges faster, but we lose the guarantee of satisfying the Bellman equations (and thus may not converge to the optimal policy).

Another approach is off-policy:

$$\frac{\partial \mathcal{L}}{\partial w_{s,a}} \approx Q(s, a; \mathbf{w}) - [r + \gamma \max_{a' \in A} Q(s', a'; \mathbf{w})] \tag{16}$$

This is know as the Q-learning update. This uses State-Action-Reward-State from the environment. Since we take a max over actions a' for the gradient update, we are going 'off-policy.' This takes longer to converge, but our update is consistent with the Bellman equations.

To summarize, no matter what our gradient estimate is, we make the gradient update

$$w_{s,a} \leftarrow w_{s,a} - \eta \frac{\partial \mathcal{L}}{\partial w_{s,a}} \tag{17}$$

For the case of Q-learning, this can be written as

$$w_{s,a} \leftarrow w_{s,a} - \eta \left(w_{s,a} - [r + \gamma \max_{a' \in A} Q(s', a'; \mathbf{w})] \right) \tag{18}$$

$$= (1 - \eta) w_{s,a} + \eta [r + \gamma \max_{a' \in A} Q(s', a'; \mathbf{w})] \tag{19}$$

Essentially, we are interpolating the old value of $w_{s,a}$ with an estimate of the future value.

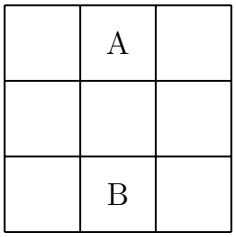
Similarly, for SARSA, the update is

$$w_{s,a} \leftarrow w_{s,a} - \eta \left(w_{s,a} - [r + \gamma Q(s', \pi(s'); \mathbf{w})] \right) \tag{20}$$

$$= (1 - \eta) w_{s,a} + \eta [r + \gamma Q(s', \pi(s'); \mathbf{w})] \tag{21}$$

3 Exercises

(Sutton & Barto 2012) Consider an MDP on the following grid:



At each square, we can go left, right, up, or down. Normally we get a reward of 0 from moving, but if we attempt to move off the grid, we get a reward of -1 and stay where we are. Also, if we move onto square A, we get a reward of 10 and are teleported to square B.

Suppose our actions also fail with probability 0.5, i.e. with probability 0.5 we stay on the current square.

Suppose our MDP is infinite horizon, and take $\gamma = 0.9$ to be the discount factor.

Defining the MDP Identify the states S, actions A, rewards, and transition probabilities $p(s'|s, a)$ in this problem.

Policy Evaluation Suppose π is the policy where we always choose to go right. Write the equations to find the values $V^{\pi}(s)$.

Value Iteration Write the second iteration of value iteration, i.e. starting by initializing $V(s) = \max_{a \in A} r(s, a)$.

Policy Iteration Write the first iteration of policy iteration, starting with $V^{\pi}(s) = 0$ for all s. (We could also initialize a policy, and do the Evaluation step to get started.)

Q-learning Suppose we are in the reinforcement learning setting where we don't know the transition probabilities $p(s'|s,a)$ or the reward function. Suppose we start at the top left square and follow an ϵ-greedy policy. At the beginning, suppose $Q(s,a) = 0$ for all s, a, except we know that we shouldn't go off the grid, so that the values of Q for the corresponding s, a pairs are -1. Take learning rate $\eta = 0.1$.

For the first step, suppose our policy tells us to explore, and we end up going right as our action (which succeeds). Write the Q-learning update.

Now write the SARSA update, assuming that our policy gives us a second action telling us to explore and go down (which succeeds).