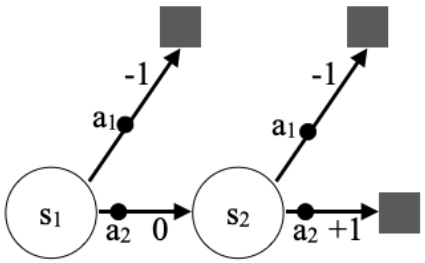

Figure 1: Example MDP

Chapter 5: From tabular learning to approximation

5.1 Off-policy TD

Consider the MDP in Figure 1. Consider a uniform behavior policy b (probability of a_1 and a_2 is 0.5 in both states). Additionally, consider the target policy π which takes a_1 with probability 0.1 and a_2 with probability 0.9 in both states. We consider the undiscounted case ($\gamma = 1$).

1. What are the Q functions Q^b and Q^π under both policies?

2. Now consider a dataset gathered using b $(s_1, a_2, 0, s_2, a_1, -1), (s_1, a_2, 0, s_2, a_2, +1)$. Consider a Q-function that is initialized as per the following table

	a_1	a_2
s_1	-1	0.5
s_2	-1	+1

   Apply one pass of Sarsa on the dataset with a learning rate of 0.1. How does the change in Q function relate to the two functions you calculated in sub-question 1?
   hint: throughout this question, only $Q(s_1, a_2)$ will change. Why?

3. A possibility for off-policy learning is applying Sarsa with importance weight. Again do one pass and notice the change in Q-function in relation to the two functions in sub-question 1.

4. Why is Q-learning considered an off-policy control method? (ex. 6.11 in RL:AI)

5. If the target policy is the greedy policy, we could use either Q-learning or importance-weighted SARSA. What would be a main difference in outcome, and which should we prefer?

6. We could also learn a V function, e.g. through TD(0), off policy. For example, by using importance weights. Why do you think the book doesn't cover this?

7. Could you do something like Q-learning for learning a V function off-policy? If yes, apply one pass. If not, explain why this is the case.

Solution

1. To calculate the Q functions, perform value iteration. It is the easiest to start at the terminal states and work your way back. For either Q function, we immediately find $Q(s_1, a_1) = -1, Q(s_2, a_1) = -1, Q(s_2, a_2) = +1$. Now, we can calculate $Q(s_1, a_2)$ using the Bellman operator:

$$Q^b(s_1, a_2) = 0.5Q(s_2, a_1) + 0.5Q(s_2, a_2) = 0$$

$$Q^\pi(s_1, a_2) = 0.1Q(s_2, a_1) + 0.9Q(s_2, a_2) = 0.8$$

2. Let's start with the hint. $Q(s_1, a_1), Q(s_2, a_1), Q(s_2, a_2)$ will have targets that are equal to their current values, so they will not change. So we only look at $Q(s_1, a_2)$.

   The first trajectory results in an update with a target of $r + Q(s_2, a_1) = -1$. So the update is $0.1(-1 - 0.5) = -0.15$, bringing $Q(s_1, a_2)$ to $0.5 - 0.15 = 0.35$.

   The second trajectory has a target of $r + Q(s_2, a_2) = +1$. The second update will thus be $0.1(1 - 0.35) = 0.065$, bringing $Q(s_1, a_2)$ to $0.35 + 0.065 = 0.415$.

   In aggregate, this on-policy update changed the Q-function in the direction of Q^b. Repeated passes would converge Q^b.

3. In the first update we have a target of -1 like in the on-policy case, but with an importance weight of $\pi(a_1|s_2)/b(a_1|s_2) = 0.2$. This results in an update of $0.1*0.2*(-1-0.5) = -0.03$, so $Q(s_1, a_2) = 0.5 - 0.03 = 0.47$.

   In the second update, we have a target of $+1$, with an importance weight of $\pi(a_2|s_2)/b(a_2|s_2) = 1.8$. This results in an update of $0.1*1.8*(1-0.47) \approx 0.095$, so $Q(s_1, a_2) \leftarrow 0.47+0.095 = 0.565$.

   In aggregate, this off-policy update changed the Q-function in the direction of Q^π. Repeated passes would converge to Q^π.

4. (ex. 6.11 book). Because the target policy and the behavior policy are different. In Q-learning the target policy is always greedy w.r.t its current values. However, its behavior policy can be anything e.g. epsilon greedy and continues to visit all state-action pairs during learning.

5. Both algorithms in principle converge to the same Q function. However, in importance-weighted SARSA if the behavior policy chooses a different action then the greedy policy, the sample is not used at all, which is a bit wasteful. And when the behavior policy chooses the same action, the importance weight is larger than one, which can make learning unstable unless a small learning rate is set (in which case learning becomes slow). So Q-learning is the preferred option.

6. This is certainly possible. However, in policy evaluation, we are often interested in evaluating the behavior policy, so off-policy learning is not as relevant. In control, where we are learning a policy, policies change over time, and we have the exploration vs. exploitation issue. Both of these issues make off-policy learning much more important.

7. For the update of $V(s_1)$, we would need to take the maximum over the 'targets' from all possible actions. However, this depends on the state reached by each such action - we typically do not have that information for all (s, a) pairs. In this particular example, the dataset doesn't have any information about the result of applying a_1 from this state, for example.

Similarly, we cannot apply a Q-learning like algorithm to Monte-Carlo learning: it requires data about the effects of trying actions that we typically do not have access to.

5.2 Function approximation and state distribution

1. Consider the state distribution, $\mu(s)$. How does it depend on the parameters of the value function approximator if we update the policy to e.g. the ϵ-greedy one?
2. How does this differ from the data distribution in standard (un-)supervised learning problems?
3. What does this mean for the weighting of the errors (such as in e.g. Eq. 9.1)?

Solution

1. $\mu(s)$ is dependent on the policy, which is controlled by the value function approximator. Thus, when the parameters change, the policy changes and so $\mu(s)$ does too.
2. In standard ML, the data distribution is independent of the learned parameters (e.g., in an image classification task, the type of images you encounter do not depend on the classifier learned so far). In RL, the states encounter do depend on the current policy.
3. While we change the parameters, we also change $\mu(s)$ and thus which states contribute most to the error we care about.