CS 181 Spring 2022 Section 10:
Reinforcement Learning

1 Introduction

In the reinforcement learning setting, we don't have direct 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 From Planning to Reinforcement Learning

Recall that MDPs are defined by a set of states, actions, rewards, and transition probabilities $\{S, A, r, p\}$, and our goal is to find the policy π^* that maximizes the expected sum of discounted rewards.

In planning, we are explicitly provided with the model of the environment, whereas in reinforcement learning, an agent does not have a model of the environment to begin with. Instead, it must interact with the environment to learn what its policy should be.

2.1 Concept Question

Would the following be problems more likely to be solved with MDP planning and which through reinforcement learning?

- Finding the best route to take through a treacherous forest using a map.
- Bringing the new Boston Dynamics robot into a new obstacle course it has never seen before.

3 Model-based Learning

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

3.1 Concept Question

Can you think of a way to do this in practice? What are some downsides?

4 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. This makes model-free learning cheaper and simpler.

To do this, we look to learn the optimal Q-values, defined as

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)V^*(s'),\ \forall s,a \tag{1}$$

where $V^*(s)$ is the optimal value function. The value $Q^*(s,a)$ is the value from taking action a in state s and then following the optimal continuation from the next state.

By learning this Q-value function, Q^*, we also have the optimal policy, with

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{2}$$

To learn Q^* we can perform a one-step decomposition, and we get an alternate form of the Bellman equations which states that for an optimal policy π^*,

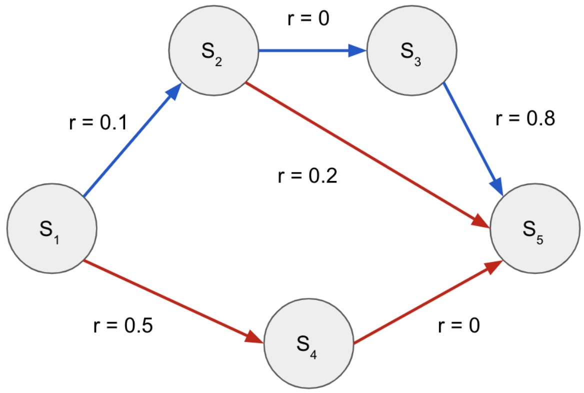
$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A}[Q^*(s',a')],\ \forall s,a \tag{3}$$

Intuitively, for an optimal policy, the Q value at the current state and action should be equivalent to the current reward plus the maximum possible expected future value from the next state.

The question then becomes how we can find the Q values that satisfy the Bellman equations as written in Equation 3. There are two ways that we do this. One is "on-policy" (SARSA) and one is "off-policy" (Q-learning).

4.1 Q values Example

In the above figure, suppose that the discount factor is 1.0 and the actions are deterministic. The agent starts from state s_1 and will finish in the state s_5, from which no more actions are taken. We can represent Q values as a table of states and actions, as shown to the right. For these converged Q values (corresponding to an optimal policy), we can quickly verify that the Q value of a given state and action is the sum of the current reward and the maximum of the Q value from the next state. We can read the optimal policy off the table by selecting the action that maximizes the value of a given state. For example, in state 1, we should take the blue action.



Q* values

	Red action	Blue action
s_1	0.5	0.9
s_2	0.2	0.8
s_3		0.8
s_4	0	
s_5		

$$Q^*(s_1, a_{blue}) = r(s_1, a_{blue}) + \max_a[Q^*(s_2, a)] = 0.1 + \max(0.2, 0.8) = 0.9$$

4.2 Exploration vs. Exploitation

An RL agent also needs to decide how to act in the environment while collecting observations. This gets to the key issue of exploration vs. exploitation.

In an exploitative approach, when we are in state s, we can simply take action $a = \arg\max_{a \in A} Q(s, a)$ based on our current estimate of the Q-function.

In an explorative approach, we want to ensure that we have visited enough states and taken enough actions from those states to get good Q-function estimates, and this can lead us to prefer to add some randomization to the behavior of the agent.

4.2.1 Concept Question

What would be a problem if our approach was only exploitative? In practice, how might we balance exploitation vs exploration?

5 On-Policy RL: SARSA

Whatever the behavior of the RL agent, a first way to update the Q-values is an on-policy method.

Given current state s, current action a, reward r, next state s', next action a' (s, a, r, s', a, hence the name SARSA), the update is

$$Q(s, a) \leftarrow Q(s, a) + \alpha_t \left[r + \gamma Q(s', a') - Q(s, a) \right] \tag{4}$$

This is known as the SARSA update (State-Action-Reward-State-Action), since we look ahead to get the action $\pi(s')$. Here, α_t, with $0 \leq \alpha_t < 1$ is the learning rate at update t. γ is the discount factor.

Here, we are taking the difference between our current Q-value at a state-action and the one that we predict using the current reward and the discounted

Q-value following the policy. We update the $Q(s, a)$ value in the direction of this difference, which is known as the "temporal difference error."

Since we follow π in choosing action a', this gradient method is on-policy. In particular, it learns Q-values that correspond to the behavior of the agent. The SARSA update rule is like we're doing a stochastic gradient descent for one observation, looking to improve our estimate of $Q(s, a)$.

Because SARSA is on-policy, it is not guranteed to converge to the optimal Q-values. In order to converge to the optimal Q-values, SARSA needs (stated informally):

- Visit every action in every state infinitely often
- Decay the learning rate over time, but not too quickly.[1]
- Move from ϵ-greedy to greedy over time, so that in the limit the policy is greedy; e.g., it can be useful to set ϵ for a state s to $c/N(s)$ where $N(s)$ is the number of times the state has been visited.

5.1 Concept Question

What would SARSA learn when following a fixed policy π? What is the tension in reducing exploration ϵ in ϵ-greedy when using SARSA to learn the optimal Q^* values?

6 Off-Policy RL: Q-Learning

Whatever the behavior of the RL agent, a second way to update the Q-values is an off-policy method.

Given current state s, current action a, reward r, next state s', the update in Q-learning is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha_t[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \tag{5}$$

Q-learning uses State-Action-Reward-State from the environment. It is the max over actions a' that makes this an 'off-policy' method. Here, we are taking the difference between our current Q-value for a state-action and the one that we predict using the current reward and the discounted Q-value when following the best action from the next state. We update the $Q(s, a)$ value to reduce this difference, which is known as the "temporal difference error."

We can see that the Q-learning update can be viewed as a stochastic gradient descent for one observation, looking to find estimates of Q-values that better approximate the Bellman equations.

[1] For each (s, a) pair, we need $\sum_t \alpha_t = \infty$ for the periods t in which we update $Q(s, a)$ (don't reduce learning rate too quickly), and $\sum_t \alpha_t^2 < \infty$ (eventually learning rate becomes small). A typical choice is to set the learning rate α_t for an update on (s, a) to $1/N(s, a)$ where $N(s, a)$ is the number of times action a is taken in state s.

Because Q-learning is off policy, it is guaranteed to converge to the optimal Q-values as long as the following is true (stated informally):

- Visit every action in every state infinitely often
- Decay the learning rate over time, but not too quickly.[2]

6.1 Concept Question

Is the Q-learning update equal to the SARSA learning update in the case that the behavior in SARSA is greedy and not ϵ-greedy?

[2]For each (s, a) pair, we need $\sum_t \alpha_t = \infty$ for the periods t in which we update $Q(s, a)$ (don't reduce learning rate too quickly), and $\sum_t \alpha_t^2 < \infty$ (eventually learning rate becomes small). A typical choice is to set the learning rate α_t for an update on (s, a) to $1/N(s, a)$ where $N(s, a)$ is the number of times action a is taken in state s.

7 Exercise: Model-free RL

Consider the same MDP below on the following grid.

	A	
	B	

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. The discount factor is $\gamma = 0.9$.

Suppose an RL agent starts at the top left square, $(0, 0)$, 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 (ie. moving off the grid from position $(0, 0)$ to $(-1, 0)$ is disallowed and corresponds with an initialization of $Q(s, a) = -1$).

The learning rate $\alpha = 0.1$. With RL, the realized reward for an action will depend on the state, the action, and whether or not the action succeeds.

1. For the first step, suppose ϵ-greedy tells the agent to explore, and the agent selects right as its action (and this action succeeds). Write the Q-learning update in step one.
2. Now write the SARSA update for step one, assuming that in addition to right in step one, ϵ-greedy tells the agent to explore and go down in the second step (and this action succeeds).
3. Are the updates the same? If not, why not?

8 TD update intuition (optional)

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:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\mathbb{E}_{s,a}\left(Q(s, a; \mathbf{w}) - [r(s, a) + \gamma \sum_{s'} p(s'|s, a) \max_{a'} Q(s', a'; \mathbf{w})]\right)^2 \quad (6)$$

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

$$\frac{\partial \mathcal{L}}{\partial w_{s,a}} = Q(s, a; \mathbf{w}) - [r(s, a) + \gamma \sum_{s'} p(s'|s, a) \max_{a'} Q(s', a'; \mathbf{w})] \tag{7}$$

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

$$\frac{\partial \mathcal{L}}{\partial w_{s,a}} \approx Q(s, a; \mathbf{w}) - [r + \gamma \max_{a'} Q(s', a'; \mathbf{w})] \tag{8}$$

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

$$w_{s,a} \leftarrow w_{s,a} - \eta \frac{\partial \mathcal{L}}{\partial w_{s,a}} \tag{9}$$

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