CS 1810 Spring 2025 Section 10 Notes:
Reinforcement Learning

1 Introduction

In the reinforcement learning setting, unlike MDPs, 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. Reinforcement learning has wide-ranging applications, from robotics to mobile health (mHealth).

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?

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

Solution

1. MDP Planning, as we are given the $\{S, A, r, p\}$ formulations through the map.
2. Reinforcement Learning, since we initialize start off with little to no knowledge about the environment, and therefore cannot solve some pre-defined MDP.

3 Model-based Learning

For model-based learning, we estimate the missing 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 construct these missing models? What are some downsides?

Solution

- One way to do this is to perform a number of random walks (at each state, we take a random action, and we track the rewards we get and the states we transition to).

  For the transition model, we maintain counts on different next states given (s, a) pairs, and we use this to update a Dirichlet model for the distribution on $p(\cdot|s, a)$ for each $s, a)$. For the reward model, one simple approach would be to track the average reward achieved for each state-action pair.

  We can then use the transition and reward models for planning. We would act according to the obtained plan, perhaps adding some ϵ-greedy exploration. This will give us more environmental interaction data, and we can use it to update our reward and transition models.

- A downside to model-based RL can be that it's unnecessarily complex— suppose there are many states, in which case the transition model $p(s'|s, a)$ becomes huge, and it may be hard to get a good estimate of the true model. This can make model-based learning inefficient and difficult to generalize.

  Also, recall that the goal is not to learn the transition model. Rather, it is to learn an optimal policy, and so we may be doing unnecessary work.

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

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

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 policy from the next state.

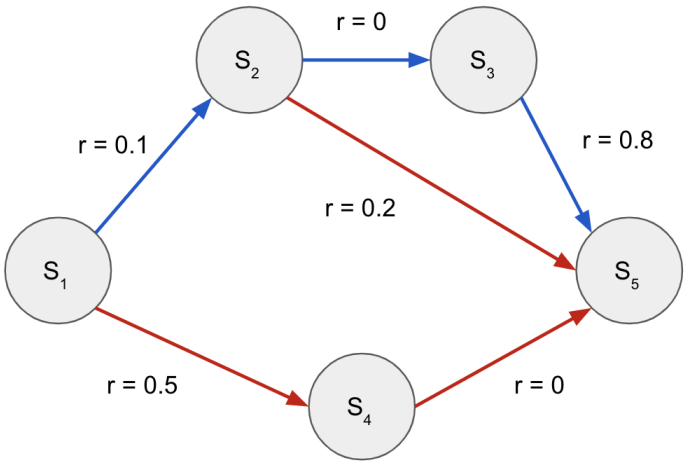
By learning this Q-value function, Q^*, we also have the optimal policy, with

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s, a)$$

We can get an alternate form by replacing $V^*(s')$ with $\max_{a' \in A}[Q^*(s', a')]$. This assumes that we follow the optimal policy from the next state.

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

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 above. Later, we will discuss two common ways to do this: "on-policy" (SARSA) and "off-policy" (Q-learning) algorithms.



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		

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

4.1 Q Values Example

In the above figure, suppose that the discount factor is $\gamma = 1$ 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, satisfying the Bellman equations, as shown above for state s_1 and action a_{blue}. We can read the optimal policy off the table by selecting the action that maximizes the Q value of a given state. For example, in state 1, we should take the blue action.

4.2 Exploration vs. Exploitation

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

- In an exploitative approach, when we are in state s, we can take action $a = \arg\max_{a \in A} Q(s, a)$, which is optimal 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, rather than taking the greedy approach.

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?

Solution

- If we focus on exploitation and do not explore enough, we might end up with a bad estimate of the Q-function and never find the true best action. We want to balance these two—exploitation is a policy that is following the advice of the Q-value estimates and useful when this knowledge is good, while exploration is necessary to properly understand the MDP. Referring back to example 4.1, we see that if we had not sufficiently explored from state 1, then the red action seems to be better to take than the blue action, which actually results in the higher overall value.
- ϵ-greedy is a sensible approach for allowing an RL agent to explore and exploit while learning in an unknown environment. In this method, we take the greedy action $\arg\max_{a \in A} Q(s, a)$ with probability $1 - \epsilon$, and pick $a \in A$ uniformly at random with probability ϵ (for some $\epsilon > 0$).

5 On-Policy RL: SARSA

Whatever the behavior of the RL agent, a first way to learn 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

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

This is known as the SARSA update (State-Action-Reward-State-Action), since we look ahead to get the next action $\pi(s') = a'$. a' is chosen thorugh the ϵ-greedy method. 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, $Q(s, a)$ and the one that we predict using the current reward and the discounted Q-value following the policy, $r + \gamma Q(s', a')$. We update the $Q(s, a)$ value in the direction of this difference, which is known as the "temporal difference error" (TD error). Intuitively, we want to update the current $Q(s, a)$ value to be closer to the sum of the reward and the Q value of the next state-action pair, in the spirit of the Bellman equations.

Since we follow π in choosing action a', this gradient method is "on-policy". In particular, at each step, it learns Q-values that correspond to the behavior of the agent. However, after each step, there will be an updated policy. The SARSA update rule is like we're doing a stochastic gradient descent for one observation, looking to improve our estimate of $Q(s, a)$ with respect to the current policy.

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 to satisfy the following properties (stated informally):

- Visit every action in every state infinitely often,
- Decay the learning rate over time, but not too quickly.[1]

[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.

- 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 if the policy π were fixed? What is the tension in reducing exploration ϵ in ϵ-greedy when trying to satisfy the convergence conditions above?

Solution

- The Q-values corresponding to policy π, i.e.

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) Q^{\pi}(s', \pi(s')).$$

- The tension with SARSA is that we need to both visit every action in every state infinitely often (explore) but also eventually stop adding random exploration (exploit). This is often called GLIE "greedy in the limit of infinite exploration."

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:

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

Q-learning uses State-Action-Reward-State from the environment. It is the max over actions a' that makes this an "off-policy" method. Unlike SARSA, the a' is not derived from some policy. Instead, it just takes a greedy approach and assume that the a' corresponding to the maximum Q-value is chosen. Again, we are taking the difference between our current Q-value for a state-action, $Q(s,a)$, and the one that we predict using the current reward and the discounted Q-value when following the best action from the next state, $r + \gamma \max_{a'} Q(s',a')$. We update the $Q(s,a)$ value to reduce this "temporal difference error."

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, like in SARSA.

Note that we do not need the last convergence condition anymore because Q-learning is off-policy. Thus, the update is independent of the choice of ϵ, since it always chooses the best action greedily.

6.1 Concept Question

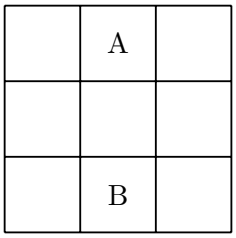
Is the Q-learning update equal to the SARSA learning update in the case that the behavior in SARSA is greedy and not ϵ-greedy?

Solution

Yes! In this case, we can easily check that the two update equations are equivalent because the next state action used in the SARSA update is $\max_{a'} Q(s', a')$, which is the same as in the Q-learning update.

7 Exercise: Model-free RL

Consider the same MDP below on the following grid.



At each square, we can go left, right, up, or down, so $A = \{\text{L}, \text{R}, \text{U}, \text{D}\}$. 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 $s \in S, a \in A$, 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 (i.e. moving off the grid from position $(0, 0)$ to $(-1, 0)$ is disallowed and corresponds with an initialization of $Q(s, a) = -1$, since the reward is -1 and it lands you in the same spot, which has initialized Q-value of 0).

Let the learning rate $\alpha = 0.1$. 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?

Solution

1. Our environment gives us $r = 10$ for succeeding with the right action, and $s' = (2, 1)$ for the next state. For Q-learning, we can use this first step and next state to update the value of $Q(s, a)$ for $s = (0, 0)$, action $a = \text{R}$, and next state $s' = (2, 1)$ as follows.

$$\begin{aligned} Q(s, a) &\leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \\ &= 0 + (0.1)(10 + 0.9 \max\{0, -1, 0, 0\} - 0) = 1 \end{aligned}$$

2. With knowledge that in next state $s' = (2, 1)$ the action $a' = \mathrm{D}$, the SARSA method will update the Q-value for state $s = (0, 0)$ and action $a = \mathrm{R}$ as follows.

$$\begin{aligned} Q(s, a) &\leftarrow Q(s, a) + \alpha(r + Q(s', a') - Q(s, a)) \\ &= 0 + (0.1)(10 + (0.9)(-1) - 0) = 0.1(10 - 0.9) = 0.91. \end{aligned}$$

3. This is a slight departure from the update done by Q-learning, and reflects the "on-policy" nature of SARSA, since in this case the action in next state s' was a result of ϵ-exploration and not a greedy choice.