Markov Decision Processes and Reinforcement Learning Review

1 Markov Decision Processes

1.1 Definition

A Markov Decision Process is defined by the following four objects:

- S: the set of states
- A: the set of actions
- P: the transition probabilities, where $P(s'|s,a)$ is the probability of transitioning to state s' upon taking action a at state s (you may also see this written as $T(s'|s,a)$).
- R: the reward function, where $R(s,a,s')$ is the reward of transitioning to s' upon taking action a at state s. Note: if the reward only depends on a subset of these quantities you may see it in forms such as $R(s,a)$ or even $R(s)$.

The agent may also have a discount factor $\gamma \in [0,1)$, which means that the rewards at time t are worth less by a factor of γ^t.

1.2 Goal

Our goal is to maximize the expected reward of the traveling agent. Therefore, we wish to learn some optimal policy $\pi^*(s)$ which tells us which action the agent should take at each state s. The policy can be deterministic or probabilistic, but we will only consider determinstic policies here.

If we know (S, A, P, R), then we can run an algorithm to compute the optimal policy $\pi^*(s)$. This is the **planning** task, since we can plan out π^* even before we take the first action. In the next section (reinforcement learning) the P, R is unknown so we must spend time transitioning around the space before we can compute π^*. The rest of this section assumes that (S, A, P, R) is known and will summarize two planning algorithms (policy and value iteration) used to solve for π^*.

1.3 Value Iteration

Suppose $V_t^*(s)$ is the optimal value function at s with t time steps remaining. We can solve for this by working backwards: if the agent only has one step remaining, it will take the action that has the highest reward:

$$V_1^*(s) = \max_a R(s,a)$$

Now suppose that we have two time steps remaining - the agent should take the action where the immediate reward plus the expected discounted value of the next state (with one less time step remaining) is maximized:

$$V_{t+1}^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} p\left(s'|s,a\right) V_t^*\left(s'\right) \right]$$

The chain of computations for value iteration looks like

$$V_1^* \to V_2^* \to V_3^* \to V_4^* \to V_5^* \to \cdots$$

Finite horizon: In this case, the game ends after T timesteps, so we can stop iterating after computing V_T^*. To find $\pi_{t+1}^*(s)$, the optimal action to take at state s with $t+1$ steps remaining, we take the argmax of the formula for $V_{t+1}^*(s)$ instead of the max:

$$\pi_{t+1}^*(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in S} p\left(s'|s,a\right) V_t^*\left(s'\right) \right]$$

Infinite horizon: The optimal V^* can be thought of as the limit of the above sequence: $V^* = \lim_{t\to\infty} V_t^*$. We won't reach V^* in a finite number of iterations (it approaches asymptotically), so we pick some convergence tolerance θ and stop iterating once the value at each state after an update changes less than θ : $|V_{t+1}^*(s) - V_t^*(s)| \leq \theta$ for all states s. However, while V_t^* does not reach V^* in finite time, the policy created using V_t^* converges to π^* after a finite number of iterations. By setting a low enough convergence tolerance, we use the resulting value function V to make π^* as follows:

$$\pi^*(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in S} p\left(s'|s,a\right) V\left(s'\right) \right]$$

Note that in the infinite horizon the policy does not depend on what turn number we are on, unlike the finite horizon case.

1.4 Policy iteration

In policy iteration, we repeat a two-step process: start with some initial policy $\pi_1(s)$, use it to create the corresponding value function $V^{\pi_1}(s)$, and then use this value function to create an updated policy $\pi_2(s)$. The chain of computations looks like

$$\pi_1 \to V^{\pi_1} \to \pi_2 \to V^{\pi_2} \to \pi_3 \to V^{\pi_3} \to \cdots$$

Step 1: Computing $V^{\pi_t}(s)$ from $\pi_t(s)$ is called policy evaluation, where the function $V^{\pi_t}(s)$ tells us the expected value gained by following policy π_t starting from state s. We can compute this in two ways. First, we can do it iteratively in a similar way to value iteration, except you remove the max over a and replace a with the policy action $\pi_t(s)$. On the other hand, we can write also it in matrix notation and solve it directly using a closed-form solution:

$$V^{\pi_t} = \left(I - \gamma P^{\pi_t}\right)^{-1} R^{\pi_t}$$

Step 2: To find $\pi_{t+1}(s)$ given a value function $V^{\pi_t}(s)$, we just take the action maximizes the immediate reward plus the expected future reward:

$$\pi_{t+1}(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s'} p\left(s' \mid s,a\right) V^{\pi_t}\left(s'\right) \right]$$

In finite time, the policy will converge and reach some optimum π^*. Once this happens, the update step will not change the policy further. Since we use π_t to make V^{π_t}, this means that the value function also converges and will not receive further updates.

1.5 Comparison

One step of value iteration takes $O(|S||A|L)$, where L is the maximum number of states reachable from any states through any action. This is intuitive, since we need to compute the value function at each s, which requires finding the expected value upon taking each action a, where the expectation is computed by summing over all the reachable states s'.

One step of policy iteration takes $O(|S||A|L + |S|^3)$, where the $|S||A|L$ comes from computing $\pi(s)$ (same reason as above) and the $|S|^3$ represents the extra work done in the policy evaluation step.

Compared to value iteration, policy iteration takes longer per iteration but takes fewer iterations to find π^*. The best to use one usually depends, but in general we prefer policy iteration.

1.6 Remark on discounting

The discount factor γ can drastically change the learned policy and value function. For a small γ, the rewards that happen in the immediate future are weighted very heavily compared to the rewards that happen later in time, so the agent is likely to be very short-sighted and optimize almost exclusively around the first few turns. For larger γ, the agent doesn't care as much about when it receives the reward, so it is willing to incur penalties in the short-term if it means that it can reap even larger rewards in the long-term.

The infinite horizon will always have a γ, but it is optional in the finite horizon case.

2 Reinforcement Learning

Now suppose that we have the same MDP setup from before, except we no longer know the transition probabilities or the rewards:

- S: the set of states
- A: the set of actions
- P: **(??????)**
- R: **(??????)**

Our task is the same as before - to learn the optimal policy $\pi^*(s)$. However, we are no longer able to plan out the optimal policy using policy or value iteration, since they depended on using P and R. Therefore, we must move around the state space and learn about the environment before we can make any statements about π^*.

There are two main types of approaches to this task: model-based and model-free learning.

2.1 Model-based learning

Although R and P are unknown, we can estimate them by transitioning around the space and recording our observations. Each time we take an action and transition to a space, we can write down the reward we got and which state we transitioned to. By visiting a state s and taking action a there many times, we can use the frequency of times that s' was transitioned to as our estimate for $p(s'|s, a)$.

Once we have our estimates of R and P, we can use planning methods like policy or value iteration from the last section to solve for $\pi^*(s)$.

One downside to this method is the fact that keeping track of all these rewards/transition probabilities and performing planning is computationally expensive. Also, to get a good estimate of P, you may need to make many visits to each state and take each action there multiple times - which requires performing a lot of exploration. If all we care about π^*, then this is more work than is necessary.

However, one benefit of model-based learning comes in the case where R, P can change with time. If the rewards or transition probabilities change, then we can just update the corresponding entries in our model and do planning on the new system to get the new π^*. Without a model for the rewards or transition probabilities, this is not possible and we need to spend some time exploring again to learn the effects of the changes.

2.2 Model-free learning

In model-free learning, we don't explicity keep track of the rewards or transition probabilities. Instead, we try to learn a Q-function, which represents the "**Q**uality" of a state-action pair. More specifically, $Q(s, a)$ tells us the long-term value of taking action a at state s. The optimal Q-function Q^* obeys the following Bellman equation:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s'} p\left(s'|s, a\right) \max_{a'} Q^*(s', a')$$

This is the immediate reward gained by the action a followed by the expected reward of an optimal follow-up. With this, we can determine $\pi^*(s)$ by just looking at which action a maximizes the Q-function at that state:

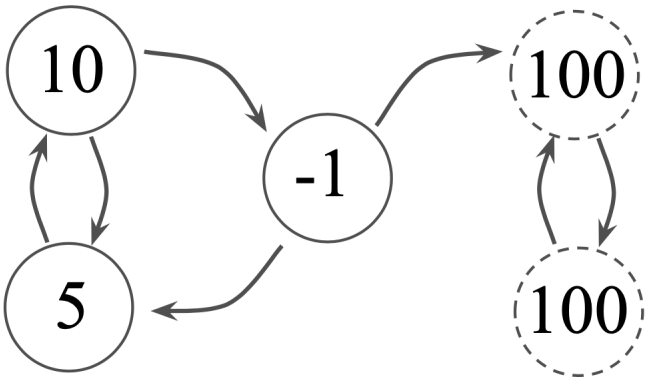
$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

If we knew what R, P were, we could easily find Q^* by doing planning and then use it to make π^*. However, given that these environment parameters are unknown, we need to estimate Q^* by transitioning around the space. Our estimate, written as $Q(s, a)$, is set to some initial values at the beginning of the task. Each time we take an action a at state s, we update $Q(s, a)$ based on what we observe. The goal is to update our $Q(s, a)$ in such a way that $Q(s, a) \to Q^*(s, a)$ as we take actions in the space. The two main ways of updating $Q(s, a)$ are called Q-learning and SARSA.

2.2.1 Movement policy

However, if we don't have π^* yet, then how should our agent move around the space? At first, the actions will essentially be arbitrary since we don't have any information on the environment. However, as we transition around the space, gain rewards, and repeatedly update the Q-table $Q(s, a)$, we get a better idea of which actions are better than others. Now the agent has a choice: To keep moving around randomly and exploring, or to use the information we have gained so far to start taking actions we believe are optimal. This is the tension between exploration and exploitation.

The purely greedy movement policy π just takes whichever action has the largest Q-value at that state. However, if our current $Q(s, a)$ is not an accurate representation of the target $Q^*(s, a)$, then the actions that we take may be locally optimal but not globally optimal. Consider the following example:



In this example, suppose the agent has only visited the states with reward 5, 10, and -1, and it has not yet visited the states with reward 100. If we act greedily, then we would transition back-and-forth between the 5 and 10 states. However, the actual best policy is to go over to the 100 states and transition back-and-forth between them. The issue is that the true quality Q^* of transitioning from the 10 to the -1 is actually very large, since being at the -1 allows us to transition to the 100. However, our agent has not experienced those rewards yet, so our estimated quality Q of transitioning from the 10 to the -1 is poor, and we would rather transition down to the 5. We can see that if the agent moves purely greedily, the agent will start exploiting before it has done enough exploring.

Instead, we can strike a balance between exploring and exploiting using the ϵ-greedy policy:

$$\pi(s) = \begin{cases} \arg\max_a Q(s,a) & \text{w.p. } 1-\epsilon \\ \text{Random} & \text{w.p. } \epsilon \end{cases}$$

This policy has us take our estimate of the optimal action with probability $1-\epsilon$, and we take a random action with probability ϵ. This way, we stay near states that have high rewards, but we also occasionally deviate to try new paths.

2.2.2 Updating $Q(s,a)$

Now that we have a policy π which tells us how to move, we need to determine a way to update our initial Q-values based on the observations we make as we transition around the space.

Suppose we are at state s, take action a to get reward r, end up at state s', and the policy tells us to take action a' as our next action. We update $Q(s,a)$ as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t[\Delta Q]$$

where α_t is the learning rate at time-step t and ΔQ is determined by which update method we are using. This looks similar to the expression for gradient descent, where adjust the parameter based on the current value plus a learning rate times a difference (temporal difference updates). The full update rules for Q-learning and SARSA are as follows:

$$\text{Q-learning: } Q(s,a) \leftarrow Q(s,a) + \alpha_t[R(s,a) + \gamma \max_{a^*} Q\left(s', a^*\right) - Q(s,a)]$$

$$\text{SARSA: } Q(s,a) \leftarrow Q(s,a) + \alpha_t[R(s,a) + \gamma Q\left(s', a'\right) - Q(s,a)]$$

2.2.3 Q-learning (off-policy)

The Q-learning update uses the quantity

$$R(s,a) + \gamma \max_{a^*} Q(s', a^*)$$

This means $Q(s,a)$ is updated based on the immediate reward obtained from action a, plus the quality of the **greedy** follow-up action at the new state s' (the move we currently think is the best). Note that the action a' the agent actually takes at s' is still determined by the ϵ-greedy policy. The term "off-policy" doesn't mean that Q-learning doesn't follow a policy - rather, it means that the update of $Q(s,a)$ does not use the a' of the next state which the policy gave us and instead picks the a with the highest Q-value.

2.2.4 SARSA (on-policy)

The SARSA update uses the quantity

$$r(s,a) + \gamma Q(s',a')$$

So $Q(s,a)$ is updated based on the reward we got from the transition, plus the quality of the **actually performed** follow-up action a' at the new state.

2.2.5 Ok, but what's the difference?

If the ϵ-greedy policy sets the follow-up move a' to be the action with the largest Q-value (greedy option), then the Q-learning and SARSA updates will be the same. However, if the policy tells us to do some random a', then SARSA will update $Q(s,a)$ based on taking the random a' as a follow-up when Q-learning will update $Q(s,a)$ based on taking the greedy action as a follow-up. Regardless, both agents will still take the random a' as their next action.

See the cliff-walking task in Sutton & Barto for a great example on the difference!

2.2.6 Convergence

Q-learning: If α_t decays at the appropriate rate and we do each (s,a) enough, then we learn the optimal Q-function: $Q \to Q^*$ as $t \to \infty$ (Q-Learning Convergence Theorem).

SARSA: The conditions for convergence are the same for Q-learning, but there is one extra: the behavior needs to be greedy in the limit as $t \to \infty$. Recall that $Q^*(s,a)$ is the reward from taking action a at state s followed by the expected reward of the optimal follow-up. However, SARSA does not always update using the optimal follow-up - it sometimes updates based on taking a random action. This is why we say SARSA does not learn Q^* - instead, it learns what we call Q^π, where $Q^\pi(s,a)$ is the quality of taking action a at state s with the understanding that afterwards we move according to the policy π.

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

In order for SARSA to learn Q^* under an ϵ-greedy policy, we need to take $\epsilon \to 0$ so that the policy is greedy in the limit and takes the optimal action each time.

2.3 Summary

Both Q-learning and SARSA are model-free reinforcement learning methods, which means that neither method learns the rewards $R(s,a)$ or transition probabilities $P(s'|s,a)$, and instead they both try to learn the optimal Q-function Q^*.

While both methods transition according to some policy π, Q-learning updates $Q(s,a)$ based on the action with the highest Q-value at s' while SARSA updates $Q(s,a)$ based on the actual follow-up action that the agent takes at s'.

2.4 Student questions:

What is the relationship between MDPs and RL?
MDPs refer to a process with states, actions, rewards, and transition proabilities, along with the Markov assumption. If we know (S, A, R, P) then we can do planning (like policy or value iteration) to find the optimal policy. Otherwise, if R, P are unknown, we must do reinforcement learning to find π^*.

What are the advantages/disadvantages of Q-learning and SARSA?
One advantage of Q-learning is that it is off-policy - we learn Q^* even with a nonzero exploration probability ϵ, which allows us to spend more time exploring. If certain parameters of the environment change with time, then Q-learning would be able to explore and learn these changes but SARSA may not since it requires $\epsilon \to 0$ where it eventually stops exploring.

One advantage of SARSA is that it is quicker to compute the update, since you just use the a' from the policy rather than taking the max over all possible action. Also, it provides risk-aversion (see the cliff-walking example in Sutton & Barto) but this can also be considered a bug. However, it may learn a suboptimal policy since exploration in SARSA needs to be greedy in the limit.

What is the relationship between the value function $V(s)$ and the Q-values $Q(s, a)$?
You can think of the Q-values as a generalization of the value function, meaning that you can make $V(s)$ from $Q(s, a)$.

$$V^*(s) = \max_a Q^*(s, a)$$

The value function $V(s)$ only takes in a state, while the Q-value tracks the reward of a state-action pair. The reason why we used $V(s)$ for planning was because we knew the transition probabilities - so to find the quality of an action a at s we can just take the weighted average $\sum_{s'} p(s'|s, a)V(s')$ to find the expected value. However, since we don't have the transition probabilities in reinforcement learning, we can't do this! So since P is unknown we need to keep track of the value of state-action pairs $Q(s, a)$ rather than just the value of states $V(s)$.

What do the Q-values have to be initialized to?
They don't have to be anything specific. You can set them all to be zero, or set them to all be very high (called optimistic initial conditions, which encourages exploration).

What does GLIE mean?
GLIE stands for "greedy in the limit of infinite exploration". It refers to the tension that SARSA faces, where in order to learn Q^* it needs to explore and visit each state infinitely often, but it also needs to be greedy in the limit and exploit what it has learned.

You need to take $\epsilon \to 0$ in SARSA, but can you do this for Q-learning as well?
Yes, you can. Note that reducing the exploring probability means you will face tension with the fact that you need to visit each state infinitely often. However, if you feel like your agent has learned well and you want to start exploiting the knowledge it has learned without making errors, then it makes sense to decay ϵ.

Answers to clear up varying degrees of confusion

Both SARSA and Q-learning initialize and update Q-values. Q-learning is off-policy, but this refers to the update rule for $Q(s, a)$ - the agent still moves according to the policy π.