Giorgio Mendoza

RBE595-S24-S04

Dr. Dadkhah Terani

Reinforcement Learning

HW2

1- Suppose $\gamma\gamma$ = 0.8 and we get the following sequence of rewards RR1 = −2,RR2 = 1, RR3 = 3, RR4 = 4, RR5 = 1.0 Calculate the value of GG0 by using the equation 3.8 (work forward) and 3.9 (work backward) and show they yield the same results

Given γ = 0.8 & Sequence of rewards R1 = -2, R2 = 1, R3 =3, R4 = 4, R5 = 1.0

Using Work forward Eq.:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$

Then G_0 *becomes*

$$G_0 = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^4 R_5$$

Using Work Backwards Eq.: $G_t = R_{t+1} + \gamma G_{t+1}$

Then we can start from the last reward and move backward calculating $G_4, G_3, G_2, G_1 and\ G_0$

```
# given values
gamma = 0.8
rewards = [-2, 1, 3, 4, 1.0]

# using Equation 3.8 (Work Forward)
G0_forward = sum([gamma**i * rewards[i] for i in range(len(rewards))])

# using Equation 3.9 (Work Backward)
G_backward = [0] * len(rewards)  # Initialize the list for G values
for t in range(len(rewards)-1, -1, -1):
    if t == len(rewards) - 1:
        G_backward[t] = rewards[t]  # For the last reward, G_t is just the reward
itself
    else:
        G_backward[t] = rewards[t] + gamma * G_backward[t+1]

G0_backward = G_backward[0]

G0_forward, G0_backward
```

```
Result: (3.177600000000001, 3.177600000000001)
```

2- Explain how a room temperature control system can be modeled as an MDP? What are the states, actions, rewards, and transitions.

To model a room temperature control system as a MDP we need to define the states, actions, rewards and transitions to capture the dynamics of temperature control.

States: The states in this MDP can represent the different possible temperatures (or temperature ranges) within the room. For simplicity, these can be discrete values, like 18°C, 19°C, 20°C, and so on, or ranges like 18-19°C, 20-21°C, etc. The state could also include other relevant factors such as the time of day or external temperature, depending on the complexity desired.

Actions: Since actions are the decisions that the control system can make at each state, then the temperature control system could include increasing the temperature, decreasing the temperature, or maintaining the current temperature. The actions could be discrete (e.g., increase by 1°C, decrease by 1°C) or continuous (e.g., set the thermostat to a specific temperature).

Rewards: Rewards are feedback signals that indicate how good an action is in a given state. For a temperature control system, the reward function could be designed to maintain comfort. For instance, the system might receive a high reward for keeping the room at an optimal temperature range and lower rewards as the temperature deviates from this range. The reward function could also penalize excessive energy use, thereby encouraging energy-efficient operation.

Transitions: Transition probabilities define the likelihood of moving from one state to another state, given an action. In a temperature control system, these probabilities would be based on how likely it is that a particular action (i.e. increasing the temperature) will result in a specific new temperature state. These probabilities can be determined from historical data or physical models of the room's thermal properties.

In summary, an MDP model for a room temperature control system would involve states representing different temperature conditions, actions that adjust the temperature, rewards based on comfort and energy efficiency, and transitions that predict the outcome of temperature adjustments. This model can then be used to develop policies (sets of rules) that dictate how the system should react in different situations to optimize comfort while minimizing energy usage.

3- What is the reward hypothesis in RL?

The reward hypothesis is a fundamental concept in Reinforcement Learning (RL) that underpins much of its theoretical and practical framework. It states:

"All of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (reward)."

This hypothesis suggests that the complex behaviors of an agent can be effectively described and motivated through a single scalar value called "reward". The agent's objective is to maximize the total amount of reward it receives over time. This approach allows for the modeling of a wide range of problems as reinforcement learning tasks.

Key points of the reward hypothesis include:

Scalar Reward: The reward is a single number (a scalar) which makes it easier to compare different states or actions. This simplification is powerful but can also be limiting since it forces the encoding of all objectives, no matter how complex, into this single number.

Cumulative Reward: The agent seeks to maximize not just immediate rewards, but the cumulative sum of rewards over time. This perspective encourages long-term planning and consideration of future consequences.

Expected Value: Due to the probabilistic nature of environments and actions in RL, the focus is on maximizing the expected value of the cumulative reward. This accounts for uncertainty and variability in outcomes.

Generality: The reward hypothesis asserts that this framework is sufficiently general to capture all types of goals and purposes. Whether the task is as simple as navigating a maze or as complex as playing a strategic game, it can be framed in terms of reward maximization.

The reward hypothesis is a very relevant topic in RL because it provides a unifying framework for a wide variety of problems. However, it also introduces challenges, such as designing an appropriate reward function that accurately reflects the desired objectives and behaviors, and dealing with the trade-off between immediate and long-term rewards.

4- We have an agent in a maze-like world. We want the agent to find the goal as soon as possible. We set the reward for reaching the goal equal to +1 With $\gamma\gamma$ = 1. But we notice that the agent does not always reach the goal as soon as possible. How can we fix this?

If an agent in a maze-like environment is not reaching the goal as quickly as possible despite a reward setup, it might indicate an issue with the reward structure or the learning algorithm itself. With the current setup, where reaching the goal yields a reward of +1 and the discount factor γ is 1, the agent receives the same total reward regardless of how long it takes to reach the goal. This setup does not incentivize the agent to reach the goal quickly. To address this, there are several approaches to be considered:

Negative Reward for Each Step: Assign a small negative reward for each step the agent takes. This way, the longer it takes to reach the goal, the more negative the cumulative reward becomes. The agent is thus incentivized to find the shortest path to the goal to minimize the negative reward.

Increasing Reward with Time: Alternatively, we can structure the rewards so that they decrease over time or steps. For instance, the agent could start with a higher potential reward which decreases with each step taken. This approach also encourages the agent to reach the goal more quickly to maximize its reward.

Time-Based Reward: Implement a time-based reward system where the reward is higher if the agent reaches the goal faster. This could be a function that decreases the reward based on the number of steps or time taken to reach the goal.

Goal Proximity Reward: Introduce intermediate rewards based on the agent's proximity to the goal. The closer the agent gets to the goal, the higher the reward. This can help guide the agent towards the goal more effectively.

Adjusting the Learning Algorithm: If the reward structure seems appropriate, but the agent still struggles, consider adjusting the learning algorithm. This could involve tuning hyperparameters, using a different learning algorithm, or employing techniques like exploration strategies or learning rate adjustments.

Penalties for Undesirable States: If there are specific states or behaviors we want to discourage (i.e. going in circles or entering certain areas), we can introduce penalties (negative rewards) for these actions. By tweaking the reward system or the learning algorithm, we can encourage the agent to learn to reach the goal as quickly as possible. It's important to carefully balance the reward structure to avoid unintended behaviors, such as the agent becoming too risk-averse or missing the goal altogether.

5- What is the difference between policy and action?

These are 2 fundamental RL concepts:

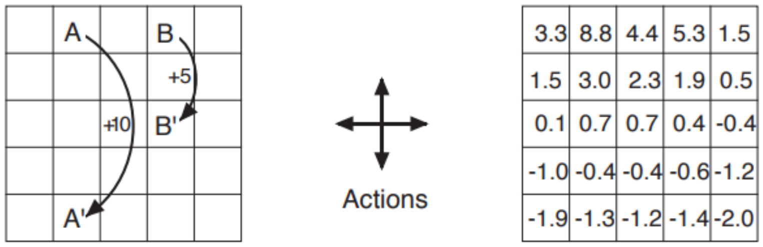
Action: An action is a specific move or decision made by the agent in a given state. It is a part of the action space, which is the set of all possible actions an agent can take in the environment. Actions are the means by which the agent interacts with and affects the environment. For example, in a game like chess, an action would be moving a specific piece to a specific square.

Policy: A policy, on the other hand, is a strategy or a rule set followed by the agent to decide which action to take in a given state. It is essentially a mapping from states of the environment to actions. The policy can be deterministic (where a state always leads to the same action) or stochastic (where actions are chosen based on a probability distribution). In RL, a policy is what the agent learns through training. The goal is to find an optimal policy that maximizes the expected cumulative reward over time. Using the chess example again, a policy would be the underlying strategy that dictates which piece to move given the current configuration of the chessboard.

So in summary, while an action is a specific decision or move made in a particular instance, a policy is the overarching strategy or set of rules that determines how an agent selects actions in various situations. The role of learning in RL is typically focused on discovering or improving policies to maximize some measure of long-term success defined by the reward structure of the task.

6- Exercise 3.14 of the textbook.

Exercise 3.14 The Bellman equation (3.14) must hold for each state for the value function v⇡ shown in Figure 3.2 (right) of Example 3.5. Show numerically that this equation holds for the center state, valued at +0.7, with respect to its four neighboring states, valued at +2.3, +0.4, 0.4, and +0.7. (These numbers are accurate only to one decimal place.)



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Figure 3.2: Gridworld example: exceptional reward dynamics (left) and state-value function for the equiprobable random policy (right).

The Bellman equation for a state *s* considering the reward and discounting is:

$$v(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$$

Where:

- *v*(*s*) is the value of state *s*,
- *π*(*a*∣*s*) is the policy, which we'll assume gives equal probability to all actions (equiprobable),
- *p*(*s*'∣*s*,*a*) is the transition probability from state *s* to state s' given action *a*,
- *r*(*s*,*a*,*s*') is the reward received when transitioning from state *s* to *s*' with action *a*,
- *γ* is the discount factor, which is 1 in this case

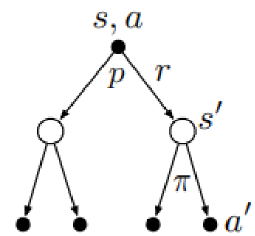
Assuming that the transitions from the central state do not have any additional rewards (as the figure does not indicate any special transitions from the central state), and that *γ*=1.
The expected value of the central state, calculated using the Bellman equation with the given values for the neighboring states and assuming an equiprobable random policy with no additional rewards for transitions from the central state, is 0.75. This does not match the stated value of 0.7 for the center state.

However, since the Bellman equation is not satisfied, this could mean that there are additional rewards or transition dynamics affecting the central state that have not been accounted for, or that the environment's dynamics are more complex than initially assumed.

The context of the grid world, such as special transitions like those from A to A' or B to B', may influence the correct calculation and must be included if they apply to the central state. If there are no special transitions affecting the central state, then the mismatch indicates that the stated value function may not correctly reflect the environment's dynamics under the current policy, or the policy itself might be different from an equiprobable one.

```
# given values for center and neighboring states from new context

center_state_value = 0.7

neighboring_states_values = [2.3, 0.4, -0.4, 0.7]

actions = 4  # number of actions is 4 (up, down, left, right)

# assuming equiprobable policy and discount factor gamma = 1

# reward for each transition is 0 except for special transitions indicated in the
figure,which do not apply to the central state.

# calculate expected value using Bellman equation

expected_center_value = sum(neighboring_states_values) / actions

# check if the expected value matches the given value for the center state

expected_center_value, expected_center_value == center_state_value

Result
(0.75, False)
```

7- Exercise 3.17 of the textbook. Exercise 3.17 What is the Bellman equation for action values, that
is, for q_π? It must give the action value q_π(s, a) in terms of the action values, q_π(s', a'), of possible successors to the state–action pair (s, a).
Hint: the backup diagram to the right corresponds to this equation.
Show the sequence of equations analogous to (3.14), but for action values.



q_π backup diagram

The Bellman equation for action values under a policy π is expressed as follows:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')]$$

- q_π (*s*,*a*) is the action-value function under policy *π*, representing the expected return of taking action *a* in state *s* and thereafter following policy *π*.
- *p*(*s*',*r*|*s*,*a*) is the probability of transitioning to state *s*' with reward *r* after taking action *a* in state *s*.
- *γ* is the discount factor, which weighs the importance of future rewards.
- $\sum_a$'*π*(*a*'|*s*') is the sum over all possible actions *a*' from the next state *s*', weighted by how likely each action *a*' is under policy *π*.
- This equation can be used iteratively to estimate the Q-values for each state-action pair in the environment. It states that the value of taking a specific action in a given state is the expected immediate reward plus the discounted value of the subsequent state-action pairs, following the current policy.

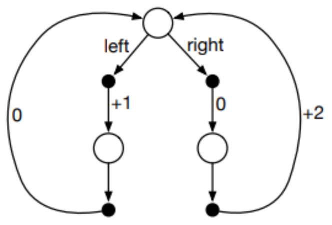
To illustrate this with a sequence analogous to the value function Bellman equation, we can update the Q-values in an iterative manner, such as in Q-learning. For each state-action pair, we update the Q-value based on the observed reward and the best estimated future value. The iterative update rule in Q-learning, which is an off-policy method, can be seen as a form of the Bellman equation:

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

Here, *α* is the learning rate, and $max_{a'}$Q(s',a') is the maximum estimated action value for the next state *s*', which represents the best expected future reward. This update rule is applied iteratively until the Q-values converge to the optimal Q-values Q^*(*s*,*a*), which reflect the maximum expected return achievable from any given state *s* by taking action *a* and thereafter following an optimal policy.

8- Exercise 3.22 of the textbook.

Exercise 3.22 Consider the continuing MDP shown on to the right. The only decision to be made is that in the top state, where two actions are available, left and right. The numbers show the rewards that are received deterministically after each action. There are exactly two deterministic policies, πleft and πright. What policy is optimal if γ = 0? If γ = 0.9? If γ = 0.5?



To solve this problem, we need to consider the rewards and the discount factor γ for the given Markov Decision Process (MDP). In the MDP, there are two actions: "left," which yields a reward of +1, and "right," which yields a reward of +2. There are two deterministic policies to evaluate: left*π* and right*π*.

- For γ = 0, the agent is myopic and only cares about immediate rewards. Hence, the optimal policy will be the one that gives the highest immediate reward.
- For γ = 0.9, the agent values future rewards as well, but they are discounted. To determine the optimal policy, we need to calculate the expected return for each policy by considering both the immediate and the future rewards.
- For γ = 0.5, the future rewards are discounted more heavily. We will again calculate the expected return for each policy, but the immediate rewards will have more weight than in the γ = 0.9 case.

The optimal policies for the given discount factors γ in the MDP are as follows:

- For γ = 0, the optimal policy is to choose "right" because the agent only cares about the immediate reward, and the immediate reward for "right" (+2) is higher than for "left" (+1).
- For γ = 0.9, the optimal policy is also to choose "right". Even though future rewards are considered, the higher immediate reward from choosing "right" results in a higher expected return when discounted over the long run.
- For γ = 0.5, the optimal policy remains to choose "right" for the same reasons as above; the immediate reward dominates the value calculation, and thus "right" with a reward of +2 is better than "left" with a reward of +1, even when future rewards are discounted quite heavily.

```
# define rewards for left and right actions
reward_left = 1
reward_right = 2
# define discount factors
gamma_values = [0, 0.9, 0.5]
```

```
# for gamma = 0, only care about immediate rewards, so optimal policy is one with the
highest immediate reward.
optimal_policy_gamma_0 = 'right' if reward_right > reward_left else 'left'
# calculate expected return for each policy for gamma = 0.9 and 0.5
# since this is a continuing MDP, consider infinite returns. Expected return is reward
divided by (1 - gamma)
# because it's the sum of geometric series: R + gamma * R + gamma^2 * R + ... = R / (1
- gamma).
expected_return_left_gamma_09 = reward_left / (1 - gamma_values[1])
expected_return_right_gamma_09 = reward_right / (1 - gamma_values[1])
expected_return_left_gamma_05 = reward_left / (1 - gamma_values[2])
expected_return_right_gamma_05 = reward_right / (1 - gamma_values[2])
# determine optimal policy for gamma = 0.9 and 0.5
optimal_policy_gamma_09 = 'right' if expected_return_right_gamma_09 >
expected_return_left_gamma_09 else 'left'
optimal_policy_gamma_05 = 'right' if expected_return_right_gamma_05 >
expected_return_left_gamma_05 else 'left'
(optimal_policy_gamma_0, optimal_policy_gamma_09, optimal_policy_gamma_05)
```

```
Result
('right', 'right', 'right')
```

Therefore, for all cases with the given rewards and the optimal policy is to always go "right" as shown above.