RL Mid Prep

March 3, 2025

Notes

Markov Decision Process (MDP)

A Markov Decision Process (MDP) is defined by the tuple (S, A, P, R, γ):

- S : Set of states
- A : Set of actions
- $P(s'|s, a)$: Transition probability from state s to s' given action a
- $R(s, a)$: Reward function defining the expected reward for taking action a in state s
- $\gamma \in [0, 1]$: Discount factor determining the importance of future rewards

Policy and Value Functions

- A policy $\pi(a|s)$ defines the probability of selecting action a in state s.
- State-value function: $v_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$
- Action-value function: $q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a\right]$

Dynamic Programming (DP)

DP methods assume a perfect model of the environment (transition probabilities known).

Bellman Equations

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s, a) \left[R(s, a) + \gamma v_\pi(s')\right]$$

Policy Evaluation and Improvement

- Policy evaluation: Iteratively update $v_\pi(s)$ using Bellman expectation equation.
- Policy improvement: Derive a better policy by selecting actions greedily:

$$\pi'(s) = \arg\max_a q_\pi(s,a)$$

- Policy iteration: Alternate between evaluation and improvement until convergence.
- Value iteration: Instead of full policy evaluation, update value function directly:

$$v(s) = \max_a \sum_{s'} P(s'|s,a) \left[R(s,a) + \gamma v(s')\right]$$

Monte Carlo Methods

- Used when the transition probabilities are unknown.
- Learn value function by averaging returns from sampled episodes.
- First-visit MC: Updates state values based on the first occurrence of a state.
- Every-visit MC: Updates state values for every occurrence of a state in an episode.

Temporal Difference (TD) Learning

TD combines Monte Carlo and DP ideas:

$$v(s) \leftarrow v(s) + \alpha \left[R + \gamma v(s') - v(s)\right]$$

- TD(0): Updates value estimates after every step.
- SARSA: On-policy TD control method.
- Q-learning: Off-policy TD control method using:

$$q(s,a) \leftarrow q(s,a) + \alpha \left[R + \gamma \max_{a'} q(s',a') - q(s,a)\right]$$

Question 1
What is the Bellman equation in the context of Reinforcement Learning?

Solution: The Bellman equation is a fundamental equation in Reinforcement Learning that expresses the value of a state as the sum of the immediate reward and the discounted value of the next state. For a state s, it is given by:

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right)$$

where:

- $V(s)$ is the value of state s,
- $R(s,a)$ is the reward for taking action a in state s,
- γ is the discount factor,
- $P(s'|s,a)$ is the transition probability to state s' given action a in state s.

Question 2
Explain the difference between on-policy and off-policy methods in Reinforcement Learning.

Solution: In Reinforcement Learning:

- **On-policy methods** learn the value of the policy that is being used to make decisions. These methods evaluate and improve the same policy that is used to select actions. Examples include SARSA and Actor-Critic methods.
- **Off-policy methods** learn the value of the optimal policy independently of the agent's actions. These methods can evaluate and improve a policy different from the one used to generate the data. Examples include Q-learning and Deep Q-Networks (DQN).

Question 3
Solve the following problems related to Markov Decision Processes (MDPs).

(a) Define a Markov Decision Process (MDP).

Solution: A Markov Decision Process (MDP) is a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. An MDP is defined by:

- A set of states S,
- A set of actions A,
- A transition function $P(s'|s,a)$,
- A reward function $R(s,a)$,
- A discount factor γ.

(b) What is the significance of the discount factor γ in an MDP?

Solution: The discount factor γ determines the importance of future rewards in an MDP. A value of γ close to 1 makes the agent prioritize long-term rewards, while a value close to 0 makes the agent focus on immediate rewards. It ensures that the total reward is finite in infinite-horizon problems.

Chapter 3 Exercises

Exercise 3.1
Devise three example tasks of your own that fit into the reinforcement learning framework, identifying for each its states, actions, and rewards. Make the three examples as different from each other as possible. The framework is abstract and flexible and can be applied in many different ways. Stretch its limits in some way in at least one of your examples.

Solution:

- **Example 1: Autonomous Drone Navigation**
  - **States**: Drone's position, velocity, and orientation.
  - **Actions**: Thrust, yaw, pitch, and roll controls.
  - **Rewards**: Positive for reaching waypoints, negative for collisions or excessive energy use.
- **Example 2: Personalized News Recommendation**
  - **States**: User's reading history and preferences.
  - **Actions**: Selecting articles to recommend.
  - **Rewards**: Positive for user engagement (clicks, reads), negative for user dissatisfaction.
- **Example 3: Climate Control in Smart Homes**
  - **States**: Indoor and outdoor temperature, humidity, and time of day.
  - **Actions**: Adjusting thermostat settings, activating humidifiers or dehumidifiers.
  - **Rewards**: Positive for maintaining comfort levels, negative for energy overuse.

Exercise 3.2
Is the reinforcement learning framework adequate to usefully represent all goal-directed learning tasks? Can you think of any clear exceptions?

Solution: The reinforcement learning framework is highly versatile but may not be suitable for all goal-directed tasks. For example, tasks requiring complex reasoning or those where the environment is not fully observable might not fit well. Additionally, tasks with non-Markovian dynamics or those requiring extensive prior knowledge might be challenging to model effectively within the standard RL framework.

Exercise 3.3
Consider the problem of driving. You could define the actions in terms of the accelerator, steering wheel, and brake, that is, where your body meets the machine. Or you could define them farther out - say, where the rubber meets the road, considering your actions to be tire torques. Or you could define them farther in - say, where your brain meets your body, the actions being muscle twitches to control your limbs. Or you could go to a really high level and say that your actions are your choices of where to drive. What is the right level, the right place to draw the line between agent and environment? On what basis is one location of the line to be preferred over another? Is there any fundamental reason for preferring one location over another, or is it a free choice?

Solution: The choice of where to draw the line between agent and environment depends on the level of abstraction desired and the specific goals of the task. A higher level of abstraction (e.g., choosing destinations) simplifies the problem but may lose important details. A lower level (e.g., muscle twitches) captures more detail but increases complexity. The preferred level should balance simplicity and relevance to the task, often guided by practical considerations and the specific requirements of the learning algorithm.

Exercise 3.4
Suppose you treated pole-balancing as an episodic task but also used discounting, with all rewards zero except for -1 upon failure. What then would the return be at each time? How does this return differ from that in the discounted, continuing formulation of this task?

Solution: In the episodic formulation with discounting, the return at each time step before failure would be zero, and upon failure, it would be -1. This differs from the continuing formulation, where the return is a sum of discounted future rewards, typically aiming to maximize the time before failure rather than just avoiding it.

Exercise 3.5
Imagine that you are designing a robot to run a maze. You decide to give it a reward of +1 for escaping from the maze and a reward of zero at all other times. The task seems to break down naturally into episodes - the successive runs through the maze - so you decide to treat it as an

episodic task, where the goal is to maximize expected total reward. After running the learning agent for a while, you find that it is showing no improvement in escaping from the maze. What is going wrong? Have you effectively communicated to the agent what you want it to achieve?

Solution: The agent may not be improving because it receives no intermediate rewards to guide its behavior towards escaping the maze. Without feedback during the maze run, the agent cannot learn which actions lead to success. To effectively communicate the goal, consider providing smaller rewards for progress towards the exit or penalties for time spent in the maze.

Exercise 3.6
Broken Vision System Imagine that you are a vision system. When you are first turned on for the day, an image floods into your camera. You can see lots of things, but not all things. You can't see objects that are occluded, and of course you can't see objects that are behind you. After seeing that first scene, do you have access to the Markov state of the environment? Suppose your camera was broken that day and you received no images at all, all day. Would you have access to the Markov state then?

Solution: After seeing the first scene, you do not have access to the full Markov state because you cannot see occluded or behind objects. If the camera is broken and you receive no images, you have no information about the environment, so you do not have access to the Markov state.

Exercise 3.8
What is the Bellman equation for action values, that is, for q_π? It must give the action value $q_\pi(s, a)$ in terms of the action values, $q_\pi(s', a')$, of possible successors to the state-action pair (s, a).

Solution: The Bellman equation for action values is:

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

Exercise 3.10
In the gridworld example, rewards are positive for goals, negative for running into the edge of the world, and zero the rest of the time. Are the signs of these rewards important, or only the intervals between them? Prove that adding a constant c to all the rewards adds a constant, v_c,

to the values of all states, and thus does not affect the relative values of any states under any policies. What is v_c in terms of c and γ?

Solution: The signs of the rewards are important as they indicate the desirability of outcomes. Adding a constant c to all rewards adds a constant $v_c = \frac{c}{1-\gamma}$ to the values of all states. This does not affect the relative values because the difference between state values remains unchanged.

Exercise 3.11
Now consider adding a constant c to all the rewards in an episodic task, such as maze running. Would this have any effect, or would it leave the task unchanged as in the continuing task above? Why or why not? Give an example.

Solution: Adding a constant c to all rewards in an episodic task would change the task because the reward is affected for each state. Consider an example where states inside the maze reduce the reward of the robot by -1, and the goal state (getting out) gives the robot a reward of $+1$. If we add a constant $c = 2$, then staying in the maze yields a reward of $+1$ on every state inside the maze, hence the robot would not be motivated to leave the maze as it can indefinitely collect rewards by staying inside.

Exercise 3.12
The value of a state depends on the values of the actions possible in that state and on how likely each action is to be taken under the current policy. We can think of this in terms of a small backup diagram rooted at the state and considering each possible action. Give the equation corresponding to this intuition and diagram for the value at the root node, $v_\pi(s)$, in terms of the value at the expected leaf node, $q_\pi(s, a)$, given $S_t = s$. This expectation depends on the policy, π. Then give a second equation in which the expected value is written out explicitly in terms of $\pi(a|s)$ such that no expected value notation appears in the equation.

Solution: The value of a state $v_\pi(s)$ is given by:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$$

The explicit form without expected value notation is:

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

Exercise 3.13
The value of an action, $q_\pi(s,a)$, depends on the expected next reward and the expected sum of the remaining rewards. Again we can think of this in terms of a small backup diagram, this one rooted at an action (state-action pair) and branching to the possible next states. Give the equation corresponding to this intuition and diagram for the action value, $q_\pi(s,a)$, in terms of the expected next reward, R_{t+1}, and the expected next state value, $v_\pi(S_{t+1})$, given that $S_t = s$ and $A_t = a$. Then give a second equation, writing out the expected value explicitly in terms of $p(s', r|s, a)$ defined by (3.6), such that no expected value notation appears in the equation.

Solution: The action value $q_\pi(s,a)$ is given by:

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

The explicit form without expected value notation is:

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

Exercise 3.14
Draw or describe the optimal state-value function for the golf example.

Solution: The optimal state-value function $v_*(s)$ for the golf example assigns to each state the maximum expected return achievable from that state. It would be highest near the hole and decrease with distance from the hole, reflecting the difficulty of reaching the hole from farther away.

Exercise 3.15
Draw or describe the contours of the optimal action-value function for putting, $q_*(s, \text{putter})$, for the golf example.

Solution: The contours of $q_*(s, \text{putter})$ would show the expected return for using the putter from different positions on the green. The values would be highest near the hole and decrease with distance, indicating the effectiveness of the putter from various locations.

Exercise 3.16
Give the Bellman equation for q_* for the recycling robot.

Solution: The Bellman equation for q_* for the recycling robot is:

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

Exercise 3.18
Give a definition of v_* in terms of q_*.

Solution: The optimal state-value function v_* is defined in terms of q_* as:

$$v_*(s) = \max_a q_*(s,a)$$

Exercise 3.19
Give a definition of q_* in terms of v_*.

Solution: The optimal action-value function q_* is defined in terms of v_* as:

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

Exercise 3.20
Give a definition of π_* in terms of q_*.

Solution: The optimal policy π_* is defined in terms of q_* as:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} q_*(s,a') \\ 0 & \text{otherwise} \end{cases}$$

Exercise 3.21
Give a definition of π_* in terms of v_*.

Solution: The optimal policy π_* is defined in terms of v_* as:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} \sum_{s',r} p(s', r|s, a') \left[r + \gamma v_*(s')\right] \\ 0 & \text{otherwise} \end{cases}$$

Chapter 4 Exercises

Exercise 4.1
If π is the equiprobable random policy, what is $q_\pi(11, \texttt{down})$? What is $q_\pi(7, \texttt{down})$?

Solution: For the equiprobable random policy:

- $q_\pi(11, \texttt{down})$ is the expected return when taking the action `down` from state 11. This value depends on the immediate reward and the value of the next state.
- $q_\pi(7, \texttt{down})$ is the expected return when taking the action `down` from state 7. This value also depends on the immediate reward and the value of the next state.

The exact values would require knowledge of the specific rewards and transition probabilities of the gridworld.

Exercise 4.2
Suppose a new state 15 is added to the gridworld just below state 13, and its actions, left, up, right, and down, take the agent to states 12, 13, 14, and 15, respectively. Assume that the transitions from the original states are unchanged. What, then, is $v_\pi(15)$ for the equiprobable random policy? Now suppose the dynamics of state 13 are also changed, such that action down from state 13 takes the agent to the new state 15. What is $v_\pi(15)$ for the equiprobable random policy in this case?

Solution:

- Initially, $v_\pi(15)$ would be the expected return from state 15 under the equiprobable random policy, considering the transitions to states 12, 13, 14, and 15.
- If the dynamics of state 13 are changed so that action down from state 13 leads to state 15, $v_\pi(15)$ would be updated to reflect the new transition probabilities and the potential for increased visits to state 15.

The exact value would depend on the specific rewards and transition probabilities.

Exercise 4.3
What are the equations analogous to (4.3), (4.4), and (4.5) for the action-value function q_π and its successive approximation by a sequence of functions $q_0, q_1, q_2, \dots$?

Solution: The equations analogous to (4.3), (4.4), and (4.5) for the action-value function q_π are:

- The Bellman equation for q_π:

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

- The iterative update for q_{k+1}:

$$q_{k+1}(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_k(s',a')\right]$$

- The convergence condition:

$$\lim_{k\to\infty} q_k(s,a) = q_\pi(s,a)$$

Exercise 4.4
In some undiscounted episodic tasks there may be policies for which eventual termination is not guaranteed. For example, in the grid problem above it is possible to go back and forth between two states forever. In a task that is otherwise perfectly sensible, $v_\pi(s)$ may be negative infinity for some policies and states, in which case the algorithm for iterative policy evaluation given in Figure 4.1 will not terminate. As a purely practical matter, how might we amend this algorithm to assure termination even in this case? Assume that eventual termination is guaranteed under the optimal policy.

Solution: To ensure termination, we can introduce a maximum number of iterations or a threshold for value changes. If the value function does not converge within the maximum iterations or if changes fall below the threshold, the algorithm can terminate. Additionally, we can detect cycles and penalize policies that lead to infinite loops, encouraging the algorithm to find policies that guarantee termination.

Exercise 4.5 (programming)
Write a program for policy iteration and re-solve Jack's car rental problem with the following

changes. One of Jack's employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one car to the second location for free. Each additional car still costs \$2, as do all cars moved in the other direction. In addition, Jack has limited parking space at each location. If more than 10 cars are kept overnight at a location (after any moving of cars), then an additional cost of \$4 must be incurred to use a second parking lot (independent of how many cars are kept there). These sorts of nonlinearities and arbitrary dynamics often occur in real problems and cannot easily be handled by optimization methods other than dynamic programming. To check your program, first replicate the results given for the original problem. If your computer is too slow for the full problem, cut all the numbers of cars in half.

Solution: This exercise requires implementing a policy iteration algorithm to solve the modified Jack's car rental problem. The solution involves:

- Defining the state space, action space, and reward function with the new constraints.
- Implementing the policy evaluation and policy improvement steps.
- Ensuring the algorithm handles the nonlinear costs and constraints correctly.

Pseudocode:

```
Input: State space $S$, action space $A$, transition probabilities $p(s', r|s, a)$, discount
       factor $\gamma$, convergence threshold $\theta$
Output: Optimal policy $\pi$ and value function $V$
Initialize $V(s)$ arbitrarily for all states $s$;
Initialize $\pi(s)$ arbitrarily for all states $s$;
repeat
  | Policy Evaluation:;
  | repeat
  |   | $\Delta \leftarrow 0$;
  |   | for each state $s \in S$ do
  |   |   | $v \leftarrow V(s)$;
  |   |   | $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) [r + \gamma V(s')]$;
  |   |   | $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
  |   | end
  | until $\Delta < \theta$;
  | Policy Improvement:;
  | policy_stable $\leftarrow$ True;
  | for each state $s \in S$ do
  |   | old_action $\leftarrow \pi(s)$;
  |   | $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$;
  |   | if old_action $\neq \pi(s)$ then
  |   |   | policy_stable $\leftarrow$ False;
  |   | end
  | end
until policy is stable;
return $\pi$, $V$
```

Algorithm 1: Policy Iteration for Jack's Car Rental Problem

Exercise 4.6
How would policy iteration be defined for action values? Give a complete algorithm for computing q_*, analogous to Figure 4.3 for computing v_*. Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book.

Solution: Policy iteration for action values involves:

- **Initialization**: Start with an arbitrary action-value function q_0.
- **Policy Evaluation**: Iteratively update q using the Bellman equation for action values until convergence:

$$q_{k+1}(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \max_{a'} q_k(s', a')\right]$$

- **Policy Improvement**: Update the policy π to be greedy with respect to the current q:
$$\pi(s) = \arg\max_a q(s, a)$$
- Repeat the evaluation and improvement steps until the policy no longer changes.

Exercise 4.7
Suppose you are restricted to considering only policies that are ϵ-*soft*, meaning that the probability of selecting each action in each state, s, is at least $\epsilon/|\mathcal{A}(s)|$. Describe qualitatively the changes that would be required in each of the steps 3, 2, and 1, in that order, of the policy iteration algorithm for v_* (Figure 4.3).

Solution: For an ϵ-soft policy:

- **Step 3 (Policy Evaluation)**: The evaluation step remains the same, but the policy being evaluated ensures that each action has at least $\epsilon/|\mathcal{A}(s)|$ probability.
- **Step 2 (Policy Improvement)**: The improvement step must ensure that the new policy remains ϵ-soft. This can be done by mixing the greedy policy with a uniform random policy.
- **Step 1 (Initialization)**: The initial policy should be ϵ-soft, ensuring that all actions have a minimum probability.

Exercise 4.8
Why does the optimal policy for the gambler's problem have such a curious form? In particular, for capital of 50 it bets it all on one flip, but for capital of 51 it does not. Why is this a good policy?

Solution: The optimal policy for the gambler's problem is designed to maximize the probability of reaching the goal capital. For capital of 50, betting it all on one flip maximizes the chance of reaching 100 in a single step. For capital of 51, a more conservative approach is taken to avoid the risk of losing and falling below 50, which would reduce the probability of eventually reaching the goal.

Exercise 4.9 (programming)
Implement value iteration for the gambler's problem and solve it for $p_h = 0.25$ and $p_h = 0.55$. In programming, you may find it convenient to introduce two dummy states corresponding to

termination with capital of 0 and 100, giving them values of 0 and 1 respectively. Show your results graphically, as in Figure 4.6. Are your results stable as $\theta \to 0$?

Solution: This exercise involves implementing value iteration for the gambler's problem and analyzing the results for different probabilities of winning a flip. The solution would include:

- Defining the state space, action space, and reward function.
- Implementing the value iteration algorithm.
- Visualizing the results and analyzing their stability as the convergence threshold θ approaches zero.

Pseudocode:

```
Input: State space $S$, action space $A$, transition probabilities $p(s', r|s, a)$, discount
       factor $\gamma$, convergence threshold $\theta$
Output: Optimal value function $V$ and policy $\pi$
Initialize $V(s)$ arbitrarily for all states $s$;
Set $V(0) \leftarrow 0$ and $V(100) \leftarrow 1$;
repeat
    $\Delta \leftarrow 0$;
    for each state $s \in S$ (except 0 and 100) do
        $v \leftarrow V(s)$;
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$;
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
    end
until $\Delta < \theta$;
Extract policy $\pi$ as $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$;
return $V$, $\pi$
```

Algorithm 2: Value Iteration for the Gambler's Problem

Exercise 4.10
What is the analog of the value iteration backup (4.10) for action values, $q_{k+1}(s, a)$?

Solution: The analog of the value iteration backup for action values is:

$$q_{k+1}(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma \max_{a'} q_k(s', a') \right]$$

Chapter 5 Exercises

Exercise 5.1
Consider the diagrams on the right in Figure 5.2. Why does the estimated value function jump up for the last two rows in the rear? Why does it drop off for the whole last row on the left? Why are the frontmost values higher in the upper diagrams than in the lower?

Solution:

- The estimated value function jumps up for the last two rows in the rear because those states are closer to the terminal state with a high reward, making them more valuable.
- It drops off for the whole last row on the left because those states are likely to lead to termination with a low reward or penalty.
- The frontmost values are higher in the upper diagrams than in the lower because the upper diagrams represent a policy that is more optimal, leading to higher expected returns.

Exercise 5.2
What is the backup diagram for Monte Carlo estimation of q_π?

Solution: The backup diagram for Monte Carlo estimation of q_π shows the state-action pair (s, a) leading to a sequence of states and actions, with the return G being the sum of rewards following the first occurrence of (s, a). The diagram emphasizes that the value of $q_\pi(s, a)$ is estimated based on the actual returns observed from (s, a) onward.

Exercise 5.3
What is the Monte Carlo estimate analogous to (5.5) for action values, given returns generated using μ?

Solution: The Monte Carlo estimate for action values $Q(s, a)$, given returns generated using μ, is:

$$Q(s, a) = \frac{\sum_{i=1}^{n} \rho_i G_i}{\sum_{i=1}^{n} \rho_i}$$

where ρ_i is the importance sampling ratio for the i-th episode, and G_i is the return following the first occurrence of (s, a) in the i-th episode.

Exercise 5.4
What is the equation analogous to (5.5) for action values $Q(s, a)$ instead of state values $V(s)$?

Solution: The equation analogous to (5.5) for action values $Q(s, a)$ is:

$$Q(s, a) = \frac{\sum_{i=1}^{n} \rho_i G_i}{\sum_{i=1}^{n} \rho_i}$$

where ρ_i is the importance sampling ratio for the i-th episode, and G_i is the return following the first occurrence of (s, a) in the i-th episode.

Exercise 5.5
In learning curves such as those shown in Figure 5.7, error generally decreases with training, as indeed happened for the ordinary importance-sampling method. But for the weighted importance-sampling method, error first increased and then decreased. Why do you think this happened?

Solution: The error for the weighted importance-sampling method first increases and then decreases because initially, the weighted estimates are influenced by a few episodes with high importance sampling ratios, leading to high variance. As more episodes are sampled, the weighted averages stabilize, reducing the error.

Exercise 5.6
The results with Example 5.5 and shown in Figure 5.8 used a first-visit MC method. Suppose that instead an every-visit MC method was used on the same problem. Would the variance of the estimator still be infinite? Why or why not?

Solution: If an every-visit MC method was used, the variance of the estimator would still be infinite. This is because the every-visit method includes multiple visits to the same state-action pair within an episode, leading to correlated samples and potentially unbounded importance sampling ratios, which can cause infinite variance.

Exercise 5.7
Modify the algorithm for first-visit MC policy evaluation (Figure 5.1) to use the incremental implementation for sample averages described in Section 2.4.

Solution: Modified Algorithm:

```
Input: Policy $\pi$, state space $S$, discount factor $\gamma$, convergence threshold $\theta$
Output: Estimated value function $V$
Initialize $V(s)$ arbitrarily for all states $s \in S$;
Initialize $N(s) \leftarrow 0$ for all states $s \in S$;
repeat
    Generate an episode using $\pi$;
    $G \leftarrow 0$;
    for each state s appearing in the episode (first-visit) do
        $G \leftarrow G + \gamma^t R_{t+1}$;
        $N(s) \leftarrow N(s) + 1$;
        $V(s) \leftarrow V(s) + \frac{G - V(s)}{N(s)}$;
    end
until convergence;
return $V$
```

Algorithm 3: First-Visit MC Policy Evaluation with Incremental Updates

Exercise 5.8
Derive the weighted-average update rule (5.7) from (5.6). Follow the pattern of the derivation of the unweighted rule (2.3).

Solution: The weighted-average update rule (5.7) is derived as follows:

$$Q_{n+1}(s,a) = \frac{\sum_{i=1}^{n} \rho_i G_i}{\sum_{i=1}^{n} \rho_i}$$

where ρ_i is the importance sampling ratio for the i-th episode, and G_i is the return following the first occurrence of (s,a) in the i-th episode.

Exercise 5.9: Racetrack (programming)
Consider driving a race car around a turn like those shown in Figure 5.11. You want to go as fast as possible, but not so fast as to run off the track. In our simplified racetrack, the car is at one of a discrete set of grid positions, the cells in the diagram. The velocity is also discrete, a number of grid cells moved horizontally and vertically per time step. The actions are increments to the velocity components. Each may be changed by $+1$, -1, or 0 in one step, for a total of nine actions. Both velocity components are restricted to be nonnegative and less than 5, and they cannot both be zero. Each episode begins in one of the randomly selected start states and ends when the car crosses the finish line. The rewards are -1 for each step that stays on the track, and -5 if the agent tries to drive off the track. Actually leaving the track is not allowed, but the position is always advanced by at least one cell along either the

horizontal or vertical axes. With these restrictions and considering only right turns, such as shown in the figure, all episodes are guaranteed to terminate, yet the optimal policy is unlikely to be excluded. To make the task more challenging, we assume that on half of the time steps the position is displaced forward or to the right by one additional cell beyond that specified by the velocity. Apply a Monte Carlo control method to this task to compute the optimal policy from each starting state. Exhibit several trajectories following the optimal policy.

Solution: Pseudocode:

```
Input: State space $S$, action space $A$, transition probabilities $p(s', r|s, a)$, discount
       factor $\gamma$, convergence threshold $\theta$
Output: Optimal policy $\pi$ and value function $V$
Initialize $Q(s, a)$ arbitrarily for all $s \in S$, $a \in A(s)$;
Initialize $\pi(s)$ arbitrarily for all $s \in S$;
repeat
    Generate an episode using $\pi$;
    for each state-action pair $(s, a)$ appearing in the episode (first-visit) do
        $G \leftarrow$ return following the first occurrence of $(s, a)$;
        $Q(s, a) \leftarrow Q(s, a) + \frac{G - Q(s,a)}{N(s,a)}$;
    end
    for each state $s$ in the episode do
        $\pi(s) \leftarrow \arg\max_a Q(s, a)$;
    end
until policy is stable;
return $\pi$, $V$
```

Algorithm 4: Monte Carlo Control for Racetrack Problem

Exercise 5.10
Modify the algorithm for off-policy Monte Carlo control (Figure 5.10) to use the idea of the truncated weighted-average estimator (5.9). Note that you will first need to convert this equation to action values.

Solution: Modified Algorithm:

```
Input: State space $S$, action space $A$, behavior policy $\mu$, target policy $\pi$, discount
        factor $\gamma$, convergence threshold $\theta$
Output: Optimal policy $\pi$ and value function $V$
Initialize $Q(s, a)$ arbitrarily for all $s \in S$, $a \in A(s)$;
Initialize $\pi(s)$ arbitrarily for all $s \in S$;
repeat
    Generate an episode using $\mu$;
    for each state-action pair $(s, a)$ appearing in the episode (first-visit) do
        $G \leftarrow$ return following the first occurrence of $(s, a)$;
        $\rho \leftarrow \prod_{t=0}^{T-1} \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$;
        $Q(s, a) \leftarrow Q(s, a) + \frac{\rho G - Q(s,a)}{N(s,a)}$;
    end
    for each state $s$ in the episode do
        $\pi(s) \leftarrow \arg\max_a Q(s, a)$;
    end
until policy is stable;
return $\pi$, $V$
```

Algorithm 5: Off-Policy Monte Carlo Control with Truncated Weighted-Average Estimator