Reinforcement Learning HW 2

Mert Bilgin (7034879)
mert.bilgin@student.uni-tuebingen.de
Lalitha Sivakumar (6300674)
lalitha.sivakumar@student.uni-tuebingen.de
Kevin Van Le (7314700)
kevin-van.le@student.uni-tuebingen.de

October 29, 2025

1 State-Action Value Function and Policy Iteration

a)

The state-action value function is given by:

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

For $q_\pi(11, \text{down})$:

$$q_\pi(11,\text{down}) = \sum_{s'} p(s' \mid 11,\text{down})\big[-1 + v_\pi(s')\big]$$

Since only $p(\text{Terminal} \mid 11,\text{down}) = 1$, we have:

$$q_\pi(11,\text{down}) = -1 + v_\pi(\text{Terminal}) = -1 + 0 = -1$$

Similarly, for $q_\pi(7,\text{down})$:

$$q_\pi(7,\text{down}) = \sum_{s'} p(s' \mid 7,\text{down})\big[-1 + v_\pi(s')\big]$$

Since only $p(11 \mid 7,\text{down}) = 1$, we have:

$$q_\pi(7,\text{down}) = -1 + (-14) = -15$$

Finally, for $q_\pi(9,\text{left})$:

$$q_\pi(9,\text{left}) = \sum_{s'} p(s' \mid 9,\text{left})\big[-1 + v_\pi(s')\big]$$

Since only $p(8 \mid 9,\text{left}) = 1$, we have:

$$q_\pi(9,\text{left}) = -1 + (-20) = -21$$

b)

We know that the state-value function under a policy π is given by:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, q_\pi(s, a)$$

The optimal value can be written in a similar fashion. That is,

$$v_*(s) = \max_\pi v_\pi(s) = \max_\pi \sum_{a \in \mathcal{A}} \pi(a \mid s)\, q_\pi(s, a)$$

For the optimal policy π_*, we have:

$$v_*(s) = \sum_{a \in \mathcal{A}} \pi_*(a \mid s)\, q_*(s, a)$$

Since the optimal policy selects the action that maximizes $q_*(s, a)$,

$$\pi_*(a \mid s) = \begin{cases} 1, & \text{if } a = \arg\max_{a \in \mathcal{A}} q_*(s, a), \\ 0, & \text{otherwise.} \end{cases}$$

Substituting this back, we obtain:

$$v_*(s) = \sum_{a \in \mathcal{A}} \left[\mathbb{I}\left(a = \arg\max_{a' \in \mathcal{A}} q_*(s, a') \right)\, q_*(s, a) \right] = \max_a q_*(s, a)$$

where we use $\mathbb{I}$ for the indicator operator.

c)

We know that for any policy π, the following holds for the action-value function:

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a\, v_\pi(s')$$

Taking the maximum over all policies, we obtain:

$$\max_\pi q_\pi(s, a) = q_*(s, a) = \max_\pi \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a\, v_\pi(s') \right)$$

Since the reward R_s^a and transition probabilities $P_{ss'}^a$ do not depend on π, we can move the maximization inside the sum:

$$q_*(s, a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a\, \left(\max_\pi v_\pi(s') \right)$$

By definition of the optimal value function $v_*(s') = \max_\pi v_\pi(s')$, we have:

$$\boxed{q_*(s, a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a\, v_*(s')}$$

d)

An optimal policy can be found by maximizing over the optimal action-value function $q_*(s, a)$:

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

It greedily selects the action with the highest estimated return according to q_*.

e)

We know that the state-value function is defined as:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, q_\pi(s, a)$$

For any policy π, the action-value function satisfies:

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a\, v_\pi(s')$$

Substituting the definition of $v_\pi(s')$ into the above equation gives:

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a \left(\sum_{a' \in \mathcal{A}} \pi(a' \mid s')\, q_\pi(s', a') \right)$$

Simplifying, we obtain the Bellman expectation equation for the action-value function:

$$\boxed{q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} P_{ss'}^a\, \pi(a' \mid s')\, q_\pi(s', a')}$$

2 Value Iteration

2.1 Gridworld: Getting started

Updated code in agent.py

2.2 Implement and test value iteration

a)

In `MazeGrid`, the start state only gets a non-zero value after 10 rounds of value iteration. This happens because value information can only propagate one step further from the goal with each iteration. Since the shortest path from the start to the goal is 10 moves, it takes 10 updates for the value to finally reach the start state.

b)

While running the policy for `BridgeGrid` with the default discount and noise parameters, the agent always avoided crossing the bridge because there was too much risk of falling off and receiving a reward of -100. Increasing the discount factor did not make the agent cross either. However, when the noise was set to 0.0, the agent started crossing the bridge to reach the high reward.

c)

Case	Discount	Noise
a	0.1	0.02
b	0.1	0.3
c	0.4	0.02
d	0.4	0.1

Table 1: Parameter settings for different cases.

Task (e) is impossible to achieve by tuning only the discount and noise. To make the agent avoid both exits and the cliff, a positive living reward would be required.

d)

On `MazeGrid` with the default parameters and 100 value iteration, we get a start–state value of 0.28. In Exercise 1, we got values of 0.00034 and 0.00221 when running 10 and 10000 episodes respectively. We see a difference between the two values because the first exercise used a random policy where all actions were equally likely. Here, we use value iteration to find the optimal policy and then calculate the value of the start state. The start state value is higher for the optimal policy because the agent is more likely to reach the goal quickly and collect the higher reward.

2.3 Custom Gridworld Analysis

The custom gridworld is defined as follows:

```
grid = [[' ',' ',' ',+10],
        [' ','#',-10,' '],
        [' ','#',-10,' '],
        [' ','#',+5,' '],
        ['S',' ',' ',' ']]
```

For discount = 0.9, noise = 0.2, livingReward = 0.0: From the start state the agent prefers taking the shortest path to the small reward of +5, but if the agent is in the top next state from the start state it prefers the longer path. The reason is the discount factor where the longer path is one step close and the shorter path one step farther away. In the bottom left corner state the agent actually prefers taking the riskier path to the big reward due to the big reward being discounted less.
For discount = 0.9, noise = 0.1, livingReward = 0.0: For this configuration where the noise is smaller, the agent takes the shortest path to the big reward. The reason is now that due to the smaller noise, the risk is now less to fall into the -10 pits so the best action is now taking that riskier path.
For discount = 0.9, noise = 0.2, livingReward = 0.5: When adding a living reward to the default configuration, the agent prefers taking the long safe path to the big reward. Interestingly, on the bottom next state to the small reward, the agent would prefer going the long path to the left to go to the big reward instead of taking the small reward by going a step up. This highlights the effect that the living reward has on the agent, where taking steps is also rewarded.
For discount = 1.0, noise = 0.2, livingReward = 0.5: Using the previous configuration with discount = 1.0 is rewarded for being in the run as long as possible. Hence the positive terminal

states becomes that states that the agent wants to avoid as it would end the run. If we assume that the top left corner is state (0,0) then the most valuable state is the state (0,2) as this state is the farthest away from all terminal states and the optimal action would be to walk into the wall. The distance to the terminal states still matter here because we have a noise factor where the agent could accidently go to a terminal state.

For discount = 0.9, noise = 0.4, livingReward = 0.0: With high noise, the agent wants to avoid the pitfall at all costs, it takes the same route as in the first configuration with a small difference when the state is the bottom right corner. Here, the agent goes to the left to avoid accidently falling into the pit.