Intro to AI Project 2: Blackjack

吴嘉骜 21307130203

2023 年 11 月 14 日

1 Problem 1: Value Iteration

Question 1a

The formula to update $V(s)$ in iteration $k+1$ is:

$$V_{k+1}(s) = \max_{a \in A(s)} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V_k(s')] = \max_{a \in A(s)} Q(s,a)$$

where $T(s,a,s')$ is the transition probability from state s to state s' under action a, and $R(s,a,s')$ is the reward of taking action a in state s and arriving at state s'. In this problem, we have $\gamma = 1$ and $V_k(2) = V_k(-2) = 0$ for all k.

In iteration $k = 0$, we just initialize every value to 0.

In iteration $k = 1$, we have:

$$Q_1(-1,-1) = 0.8 \times (20+0) + 0.2 \times (-5+0) = 15$$
$$Q_1(-1,+1) = 0.7 \times (20+0) + 0.3 \times (-5+0) = 12.5$$
$$V_1(-1) = \max\{Q_1(-1,+1), Q_1(-1,-1)\} = 15$$

$$Q_1(0,-1) = 0.8 \times (-5+0) + 0.2 \times (-5+0) = -5$$
$$Q_1(0,+1) = 0.7 \times (-5+0) + 0.3 \times (-5+0) = -5$$
$$V_1(0) = \max\{Q_1(0,+1), Q_1(0,-1)\} = -5$$

$$Q_1(1,-1) = 0.8 \times (-5+0) + 0.2 \times (100+0) = 16$$
$$Q_1(1,+1) = 0.7 \times (-5+0) + 0.3 \times (100+0) = 26.5$$
$$V_1(1) = \max\{Q_1(1,+1), Q_1(1,-1)\} = 26.5$$

In iteration $k = 2$, we have:

$$Q_2(-1,-1) = 0.8 \times (20+0) + 0.2 \times (-5-5) = 14$$
$$Q_2(-1,+1) = 0.7 \times (20+0) + 0.3 \times (-5-5) = 11$$
$$V_2(-1) = \max\{Q_2(-1,+1), Q_2(-1,-1)\} = 14$$

$$Q_2(0,-1) = 0.8 \times (-5+15) + 0.2 \times (-5+26.5) = 12.3$$
$$Q_2(0,+1) = 0.7 \times (-5+15) + 0.3 \times (-5+26.5) = 13.45$$
$$V_2(0) = \max\{Q_2(0,+1), Q_2(0,-1)\} = 13.45$$

$$Q_2(1,-1) = 0.8 \times (-5-5) + 0.2 \times (100+0) = 12$$
$$Q_2(1,+1) = 0.7 \times (-5-5) + 0.3 \times (100+0) = 23$$
$$V_2(1) = \max\{Q_2(1,+1), Q_2(1,-1)\} = 23$$

To sum up, we have the following table:

Table 1: Value iteration result(1a)

Iteration	State				
	-2	-1	0	1	2
0	0	0	0	0	0
1	0	15	-5	26.5	0
2	0	14	13.45	23	0

Question 1b

The optimal policy for some state s is defined as:

$$\pi_{opt}(s) = \arg\max_{a \in A(s)} Q(s,a)$$

where $Q(s,a)$ is the value of taking action a in state s.
Based on the second iteration, we have $\pi_{opt}(-1) = -1$, $\pi_{opt}(0) = +1$ and $\pi_{opt}(1) = +1$.

2 Problem 2: Transforming MDPs

Question 2a

The equation $V_1(start) \geq V_2(start)$ does not always hold. We have constructed a counterexample in `submission.py`, where you can refer to for more details. The basic idea is that the original settings may drive the agent to a bad state with a large probability despite choosing the optimal policy, while the noise in the new settings can increase the chance of reaching a good state, or gaining more reward.

Question 2b

For acyclic MDPs, the search tree can be viewed as a Directed Acyclic Graph (DAG). By performing a topological sort, the nodes of the graph are arranged into a linear sequence such that for every directed edge (s', a, s), state s' precedes s. To get the optimal values of s, we only need to know the optimal values of its successor states. Therefore, we can update the values of states in reverse topological order. This allows for the computation of each state's optimal value with only a single pass.
The algorithm is shown in Algorithm 1.

Algorithm 1: Value Iteration for Acyclic MDP

Data: MDP State Transition Graph G, Reward Model R, Discount Factor γ

Result: Optimal values $V(s)$ for all states s

`// Step 1: Topological Sort`

Perform topological sort on the state transition graph G ;

`// Step 2: Initialization`

Initialize $V(s) = 0$ for all states s ;

`// Step 3: Reverse Update in one pass`

for *each state s in reverse topological order* **do**

  Calculate $V(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) \left(R(s,a,s') + \gamma V(s')\right)$;

  `//` $V(s)$ `depends on the updated values of successor states, already known`

end

Question 2c

We regard o as a new termination state and for every state s (not a terminal one) in the original MDP, we add an edge $s \to o$. And we set $V(o) = 0$. As we will see later, o state is used to maintain the transition probability to be 1 in the new MDP.
The new MDP is defined as follows:

$$S' = States \cup \{o\} = S \cup \{o\},\ A' = Actions(S) = A(S),\ \gamma' = 1$$

$$T'(s,a,s') = \begin{cases} \gamma T(s,a,s') & \text{if } s' \neq o \\ 1-\gamma & \text{if } s' = o \end{cases}$$

$$R'(s,a,s') = \begin{cases} \frac{1}{\gamma} R(s,a,s') & \text{if } s' \neq o \\ 0 & \text{if } s' = o \end{cases}$$

If s is an original terminal state, the definition for its value still remains the same in the new MDP.
We prove that $V_{opt}(s)$ for all $s \in S$ are equal under the original MDP and the new MDP.

$$\begin{aligned}
V_{k+1}(s) &= \max_{a \in A(s)} \left\{ \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V_k(s')] \right\} \\
&= \max_{a \in A(s)} \left\{ \sum_{s' \in S} \gamma T(s,a,s') \left[\frac{1}{\gamma} R(s,a,s') + V_k(s') \right] \right\} \\
&= \max_{a \in A(s)} \left\{ \sum_{s' \in S} T'(s,a,s')[R'(s,a,s') + \gamma' V_k(s')] + T'(s,a,o)[R'(s,a,o) + \gamma' V_k(o)] \right\} \\
&= \max_{a \in A'(s)} \left\{ \sum_{s' \in S'} T'(s,a,s')[R'(s,a,s') + \gamma' V_k(s')] \right\} \\
&= V'_{k+1}(s)
\end{aligned}$$

As the value iteration formula for the new MDP is the same as the original one, we have $V'_{opt}(s) = V_{opt}(s)$ for all $s \in S$.

3 Problem 4: Learning to Play Blackjack

Question 4b

The results are shown in Figure 1.

```
----- START PART 4b-helper: Helper function to run Q-learning simulations for question 4b.
ValueIteration: 5 iterations
For small MDP, the match rate is: 0.9630
Number of total states is: 27; Number of diferent actions is: 1
ValueIteration: 15 iterations
For large MDP, the match rate is: 0.6787
Number of total states is: 2745; Number of diferent actions is: 882
----- END PART 4b-helper [took 0:00:03.529779 (max allowed 60 seconds), 0/0 points]
```

Figure 1: Q-learning vs value iteration (4b)

For small MDP, the performance of Q-learning and value iteration are similar. Their policy match rate is 96.3% with a total number of 27 states and only 1 differnet action appears.
However, for large MDP, they produce a lot more different actions in a much larger state space. The overall states are 2745, and the number of different actions is 882, leading to a match rate of 67.9%.
The reason is probability related to `identityfeatureextractor()` that the Q-learning algorithm does converge to the optimal policy. It simply assign 1 for every existing (s, a) pair, which plays no role in value function approximation.
Large MDP has a much larger state space. In the course of 30,000 (small in contrast to state space) simulation steps, a substantial number of the states cannot be encountered or they appear rarely, so the predicted Q-values remain 0 or not accurate at all. The Q-learning algorithm fails to learn the strategies for these states. As a result, there is a significant disparity and performance of Q-learning degrades.

Question 4d

The results are shown in Figure 2.

```
----- START PART 4d-helper: Helper function to compare rewards when simulating RL over two different MDPs in question 4d.
ValueIteration: 5 iterations
The average reward for value iteration with 30 trials is: 6.8333
The average reward for value iteration with 10000 trials is: 6.8274
The average reward for Q-learning with 30 trials is: 6.8000
The average reward for Q-learning with 10000 trials is: 9.4815
----- END PART 4d-helper [took 0:00:02.025395 (max allowed 60 seconds), 0/0 points]
```

Figure 2: OriginalMDP policy vs Q-learning (4d)

As observed from above, the average reward with value iteration provided by `originalMDP` is 6.833 for 30 trials and 6.827 for 10000 trials. In comparison, the average reward under Q-learning is 6.800 and 9.482 respectively. Q-learning produces a larger expected reward than value iteration.
Value iteration, based on the old policy from `originalMDP`, struggles in `newThresholdMDP`, where despite a raised threshold of 15, the expected reward falls below. This indicates that the original optimal policy

may not be suitable for the new settings, and transferring optimal policies between similar MDPs can be challengeable.

In contrast, Q-learning directly applied to `newThresholdMDP` initially performs poorly due to a limited number of trials. However, with more trials, Q-learning significantly improves, outperforming `FixedRLAlgorithm`. This enhancement showcases Q-learning's adaptability, enabling it to better align with the characteristics of the new MDP and achieve a higher expected reward.