# Project 3 - Blackjack

Yuze Ge 19307130176

November 2021

## 1 Problem 1: Value Iteration

**a.** We use this formula to update the value of V.

$$V_{k+1}(s) = \max_a \sum_{s'} P\left(s' \mid s, a\right) \cdot \left[R\left(s, a, s'\right) + \gamma V_k\left(s'\right)\right] = \max_a Q(s, a)$$

Under the settings of problem 1, the discount factor $\gamma$ is 1 and $V(-2) = V(2) = 0$.
In iteration 0, we don't have time to do anything, so we stay in place and the reward is always 0. $V_0(s) = 0$

In iteration 1, $V_1(-2) = V_1(2) = 0$

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

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

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

After iteration1, $V_1(-1) = 15$, $V_1(0) = -5$, $V_1(+1) = 26.5$.

In iteration2, $V_2(-2) = V_2(2) = 0$

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

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

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

| 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 |

After iteration2, $V_2(-1) = 14$, $V_2(0) = 13.45$, $V_2(+1) = 23$.
**b.** We choose the action $\pi(s)$ that maximizes the value of Q(s,a).

$$\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P\left(s' \mid s, a\right) \cdot \left[R\left(s, a, s'\right) + \gamma V_k\left(s'\right)\right]$$

We have:
$\pi_{opt}(-1) = -1$, $\pi_{opt}(0) = +1$, $\pi_{opt}(+1) = +1$.

# 2 Problem 2: Transforming MDPs

**a.** $V_1\left(s_{\text{start}}\right) \geq V_2\left(s_{\text{start}}\right)$ is not always satisfied. A counterexample is constructed in ***submission.py***. When action a is likely to result in a bad outcome, noise reduces the probability of a bad outcome and thus increases the expected reward.

**b.** We use such an algorithm: First, topologically sort the states in the MDP, and then calculate the value of $V_{opt}$ for each node in the reverse order of the sort.
According to the formula:

$$V(s) = \max_{a} \sum_{s'} P\left(s' \mid s, a\right) \cdot \left[R\left(s, a, s'\right) + \gamma V\left(s'\right)\right] = \max_{a} Q(s, a)$$

To calculate $V(s)$ we only need to calculate $V(s')$. Because the MDP is acyclic, $s'$ must appear after $s$ in a complete implementation. So we sort all the states in a topological order and calculate $V(s)$ in reverse order. When we are at the state s, all the $V(s')$ has been calculated, and we only need to traverse all the $(s, a, s')$ triples to calculate $V(s)$. In the end we calculate all the $V_{opt}(s)$ with only a single pass over all the $(s, a, s')$ triples.

**c.** We treat $o$ as a new termination state and create an edge $s \to o$ for all states s (s is not a termination state). And we set $V(o) = 0$.
Define the new transition probability and reward function:

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

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

So for $\forall s$ we have:

$$\begin{aligned} V_{k+1}(s) &= \max_a \left\{ \sum_s T(s,a,s') \left[ R(s,a,s') + \gamma V_k(s') \right] \right\} \\ &= \max_a \left\{ \sum_s \gamma T(s,a,s') \left[ \frac{1}{\gamma} R(s,a,s') + V_k(s') \right] \right\} \\ &= \max_a \left\{ \sum_{s' \neq o} T'(s,a,s') \left[ R'(s,a,s') + \gamma' V_k(s') \right] + T'(s,a,o) \left[ R'(s,a,o) + \gamma' V_k(o) \right] \right\} \\ &= \max_a \left\{ \sum_{s'} T'(s,a,s') \left[ R'(s,a,s') + \gamma' V_k(s') \right] \right\} \end{aligned}$$

$\gamma' = 1$
Therefore, the optimal values for all $s \in S$ are equal under the original MDP and the new MDP.

# 3 Problem 4: Learning to Play Blackjack

**b.** I implemented *simulate_QL_over_MDP* and the results are as follows:

```
----- START PART 4b-helper: Helper function to run Q-learning simulations for question 4b.
ValueIteration: 5 iterations
The match rate between Q-learning and value iteration in smallMDP is 0.9629629629629629.
 The total number of policy actions is 27 and 1 of them are different
ValueIteration: 15 iterations
The match rate between Q-learning and value iteration in largeMDP is 0.6207650273224044.
 The total number of policy actions is 2745 and 1041 of them are different
----- END PART 4b-helper [took 0:00:02.028904 (max allowed 60 seconds), 0/0 points]
```

In smallMDP, the policies of Qlearning and value iteration are very close. The match rate is up to 0.96 and only at 1 state the two models produces different actions. However, in largeMDP they produce different actions in a lot more states. The match rate is only 0.62.

We found that ***identityFeatureExtractor()*** is just an indicative function of state $s$ and action $a$. *identityFeatureExtractor(s,a)* returns the existence of the $((s,a)\ ,1)$ pair. This design does not take advantage of value function approximation (VFA) because the state space is very large and the weight function will have many components. In order to approximate $w_{s,a}$, we must visit $(s,a)$ in one realization.
In smallMDP, the state space is not large so that we can accurately estimate the $Q(s,a)$. In largeMDP, the state space becomes larger. If we still

simulate for 30000 times, some features $(s, a)$ may not have been visited yet or the number of visits is too few, which eventually leads to the $Q(s, a)$ not converging to the optimal value.

**d.** I implemented *compare_changed_MDP* and the results are as follows:

```
----- 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 of value iteration in modified_mdp with 100 trials is 6.86
The average reward of Q-learning in modified_mdp with 100 trials is 7.21
The average reward of value iteration in modified_mdp with 20000 trials is 6.82905
The average reward of Q-learning in modified_mdp with 20000 trials is 9.6745
----- END PART 4d-helper [took 0:00:01.749687 (max allowed 60 seconds), 0/0 points]
```

From the results, we can see that the average reward of value iteration is generally lower than Q-learning. The reward of value iteration is about 6.8 when the number of trials is 100 and 20000, while the average reward of Q-learning increases with the number of trials, from 7.2(100 trials) to 9.6 (20000 trials).

Value iteration uses the old strategy learned from original mdp. In modified mdp, it does not perform very well because the optimal strategy has changed in modified mdp. In addition, value iteration did not update its own strategy in the modified mdp.
Q-learning constantly updates its strategy during the simulation process. As the number of trials increases, Q-learning obtains more samples to better estimate the parameters of the model and finally gets high rewards.
In the modified mdp, the threshold has been raised. Q-learning can continuously update the strategy to get high rewards, while value iteration uses the previously learned model (low threshold), and the strategy adopted is more conservative, so it gets low rewards.
Therefore, even if the original mdp is slightly changed, using Q-learning can continue to learn new strategies in the new trials.