Report of Project3-Blackjack

Ziqin Luo 18307130198

November 26, 2020

1. Problem 1 Value Iteration

a) Under the settings of problem 1, the formula of Value Iteration should be in the form of the format below:

$$\boldsymbol{V_{opt}(s) = \max_{a}\{R(s,a) + \gamma Q(s,a)\} = \max_{a}\{R(s,a) + \gamma \sum_{a} P(s'|s,a)V_{opt}(s')\}}$$

To calculate the whole iterations, first we need to derive the **Reward Model** and the **Transition Model**. According to the conditions provided in the description, we have:

Reward Model:

R(s, a)	$s = -1$	$s = -1$	$s = -1$
$a = -1$	15	-5	16
$a = -1$	12.5	-5	26.5

Transition Model:

$a = -1$	$s' = -2$	$s' = -1$	$s' = 0$	$s' = 1$	$s' = 2$
$s = -2$					
$s = -1$	0.8		0.2		
$s = 0$		0.8		0.2	
$s = 1$			0.8		0.2
$s = 2$					

$a = +1$	$s' = -2$	$s' = -1$	$s' = 0$	$s' = 1$	$s' = 2$
$s = -2$					
$s = -1$	0.7		0.3		
$s = 0$		0.7		0.3	
$s = 1$			0.7		0.3
$s = 2$					

Besides, we know that $\boldsymbol{\gamma = 1}$. Thus, we can calculate the Value Iteration in the form of a table:

Iteration	$s = -2$	$s = -1$	$s = 0$	$s = +1$	$s = +2$

0	0	0	0	0	0
a = −1		15	−5	16	
a = +1		12.5	−5	26.5	
opt_action		a = −1	a = −1	a = +1	
1	0	15	−5	26.5	0
a = −1		14	12.3	12	
a = +1		11	13.45	23	
opt_action		a = −1	a = +1	a = +1	
2	0	14	13.45	23	0

Note: we always take the Values of terminal states as 0

From the table above, we know that after 0 iteration, which is equivalent to initialization, the values of each state:

$$V_{opt}(-2) = V_{opt}(2) = V_{opt}(-1) = V_{opt}(0) = V_{opt}(1) = 0$$

After the 1^{st} iteration:

$$V_{opt}(-2) = V_{opt}(2) = 0, V_{opt}(-1) = 15, V_{opt}(0) = -5, V_{opt}(1) = 26.5$$

After the 2^{nd} iteration:

$$V_{opt}(-2) = V_{opt}(2) = 0, V_{opt}(-1) = 14, V_{opt}(0) = 13.45, V_{opt}(1) = 23$$

b) We can derive the answers directly from question (a) :

$$\pi_{opt}(-1) = -1, \quad \pi_{opt}(0) = +1, \quad \pi_{opt}(1) = +1$$

2. Problem 2 Transforming MDPs

a) Answer: The inequality $\boldsymbol{V_1(s_{start}) \geq V_2(s_{start})}$ can't always holds.
The counter example I constructed in submission.py is just the same as the simple MDP in Problem 1. Please refer to the source code for more details.

b) We know that the MDP we have is acyclic. It implies that for any state $\boldsymbol{s}$, it will only appear in the Search Trees of out MDP problem at some specific layer for once. That's to say we can store all the $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'})$ triples in a list container and sort them in a topological order so that the $(\boldsymbol{s_1}, \boldsymbol{a}, \boldsymbol{s_1'})$ triple with a deeper state $\boldsymbol{s_1}$ (away from the root) comes before the $(\boldsymbol{s_2}, \boldsymbol{a}, \boldsymbol{s_2'})$ triple with a shallower state $\boldsymbol{s_2}$ (close to the root). Then, we can visit these triples successively and calculate their V_{opt}. This is equivalent to the operations that we start calculating the V_{opt} of states at the bottom of the MDP Search Tree and then calculate the V_{opt} of their father nodes, until we calculate the V_{opt} of root node in the Search Tree. It is obvious that to update the V_{opt} of all the states, we just need only a single pass over all the $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'})$ triples.

c) We know that:

$$\boldsymbol{States' = States \cup \{o\}, Actions'(s) = Actions(s), \gamma' = 1, \gamma < 1}$$

For $\forall\, s \in States$, we define set

$$S'_s = \{s' : T(s,a,s') > 0, s \text{ is } \mathbf{NOT} \text{ a terminal state}, a \in Actions(s)\}.$$

$$S'_{snew} = S'_s \cup \{\boldsymbol{o}\}$$

That is to say we see state $\boldsymbol{o}$ as a possible successor state of all states in and we treat it as a terminal state, which implies that $V_{opt}(\boldsymbol{o}) = 0$.

Then, we redefine the new Reward Model $\boldsymbol{R'(s,a,s')}$ and Transition Model $\boldsymbol{T'(s,a,s')}$ as below:

- For primal model $\boldsymbol{T(s,a,s')}$ and new model $\boldsymbol{T'(s,a,s')}$**:**
  - Let $\boldsymbol{T'(s,a,s') = \gamma T(s,a,s')}$.
  - $\forall\, s \in States$ (s not terminal), let $\boldsymbol{T'(s,a,o) = 1 - \sum_{s' \neq o} T'(s,a,s') = 1 - \gamma}$
- For primal model $\boldsymbol{R(s,a,s')}$ and new model $\boldsymbol{R'(s,a,s')}$**:**
  - Let $\boldsymbol{R'(s,a,s') = \frac{1}{\gamma} R(s,a,s')}$
  - $\forall\, s \in States$ (s not terminal) , let $\boldsymbol{R'(s,a,o) = 0}$

Then we have for $\forall\, s \in State$ and s is not a terminal state:

$$V_{opt}(s) = \max_{a \in Actions(s)} \{R(s,a) + \gamma Q(s,a)\}$$

$$= \max_{a \in Actions(s)} \left\{ R(s,a) + \gamma \sum_{s'} T(s,a,s') V_{opt}(s') \right\}$$

$$= \max_{a \in Actions(s)} \left\{ \sum_{s'} \gamma T(s,a,s') \frac{1}{\gamma} R(s,a,s') + \sum_{s'} \gamma T(s,a,s') V_{opt}(s') \right\}$$

$$= \max_{a \in Actions(s)} \begin{aligned} &\{ \sum_{s' \in S'_s} T'(s,a,s') R'(s,a,s') + T'(s,a,o) R'(s,a,o) \\ &+ \sum_{s' \in S'_s} T'(s,a,s') V_{opt}(s') + T'(s,a,o) V_{opt}(o) \} \end{aligned}$$

$$= \max_{a \in Actions'(s)} \left\{ \sum_{s' \in S'_{snew}} T'(s,a,s') R'(s,a,s') + \sum_{s' \in S'_{snew}} T(s,a,s') V_{opt}(s') \right\}$$

$$= \max_{a \in Actions'(s)} \{R'(s,a) + \mathbf{1} * Q'(s,a)\} \qquad (*)$$

From $(*)$, we know the new MDP we build is equivalent to the original MDP.

3. Problem 4: Learning to Play Blackjack

Question (b) :

To illustrate the comparison result between Q-learning policy and the policy learned by Value Iteration, we need to finish the implementation of `simulate_QL_over_MDP()` in submission.py. Please refer to the source code for more details. There are detailed comments beside the codes.

After running the function `run4bHelper()` in grader.py, we get the match rate between between Q-learning policy and the policy learned by Value Iteration which is showed below:

```
----- START PART 4b-helper: Helper function to run Q-learning simulations for question 4b.
ValueIteration: 5 iterations
The match rate of using identityFeatureExtractor for smallMDP: 0.9737
Number of different actions: 1  Number of total actions: 38
ValueIteration: 15 iterations
The match rate of using identityFeatureExtractor for largeMDP: 0.7010
Number of different actions: 882  Number of total actions: 2950
----- END PART 4b-helper [took 0:00:03.405997 (max allowed 1000 seconds), 0/0 points]
```

We can see that when we run `simulate()` on `smallMDP`, the match rate is around 97% and only 1 state they produce a different action. We run `simulate()` on `largeMDP`, however, the performance gets worse and match rate is only around 70% and 882 states they produce a different action.

The reasons why the performance degrade, can be explained by the feature extractor we use. Refer to the implementation of `identityFeatureExtractor()`, we find that this feature extractor is just an indicator for the states in State Space of MDP. This implies that only when we encounter a transition process that contains the (s, a) state-action pair in the simulation, can we update the parameter $w_{s,a}$ which are useful to get $Q(s, a)$. When the complexity of MDP grows, the size of State Space grows and this leads to more new state-action pairs appear and to evaluate their Q values we need to simulate more transition processes that contain these new state-action pairs to update the parameters $w_{s,a}$. Therefore, when we run `simulate()` on `largeMDP` but still with only 30000 trials, there will be a lot more (s, a) pairs that may not be encountered in the trials or they appeared too few. Both of the two situations lead to the insufficient updates for the parameters. This accounts for the reasons why we can't have relatively accurate evaluations for their Q values

to get the optimal policy.

Question (d) :

To better answer question (d), we need to finish the implementation of `compare_changed_MDP()` in submission.py. Please refer to the source code for more details. (For the reproductivity of this trial, I have set the random seed as 123.)

The screenshot below shows the results of this comparison trial:

```
----- START PART 4d-helper: Helper function to compare rewards when simulating RL ove
ValueIteration: 5 iterations
*** Expected return for FixedRLAlgorithm (numTrials=30): 6.8333 ***
*** Expected return for FixedRLAlgorithm (numTrials=30000): 6.8309 ***
*** Expected return for QLearningRLAlgorithm (numTrials=30): 0.7000 ***
*** Expected return for QLearningRLAlgorithm (numTrials=30000): 9.2527 ***
----- END PART 4d-helper [took 0:00:03.698559 (max allowed 60 seconds), 0/0 points]
```

As we can see, the expected reward with the policy provided by original MDP is **6.8309**(30 trials) and **6.8309**(30000 trials) while it is **0.7000**(30 trials) and **9.2527**(30000trials) when using Q-Learning.

From the results, we know two things:

- From submission.py, we know that the threshold of `newThresholdMDP` is 15 but by using optimal policy of original MDP, the expected reward is far less than threshold. This implies that the optimal policy of original MDP is too **conservative** to play blackjack under the new settings.

  This tells us that even though two MDPs maybe similar, the optimal policy of one MDP may behave badly in another one.

- Q-learning is run directly on the `newThresholdMDP`. When just do a few trials, Q-learning has too few samples to learn and its performance is very bad. However, if we conduct more trials and we find that Q-learning has improved dramatically and has a relatively better performance than the fixedPolicy does.

  This implies that Q-learning can improve itself step by step using the samples, which is equivalent to find a better policy that is more suitable for `newThresholdMDP` step by step, and finally get a higher expected reward than the fixedPolicy.

Therefore, we know that Q-learning can become better and better through trials and perform better and better while the fixedPolicy always stays the same.