Blackjack Report

Shun Zhang
School of Data Science
Fudan University
Shanghai, China
15300180012@fudan.edu.cn

Note that this is the writing part of the project on *blackjack*. Coding part please refer to *submission.py*.

Problem 1: Value Iteration Here, since the MDP problem is defined in an 'reward by action' way, which means that a certain reward is correlated with a certain action, our value iteration will base on the following formula:

$$U'(s) = max_{a \in A(s)} \{ \sum_{s' \in S(s,a)} P(s'|s,a) \big[\gamma * U(s') + R(s,a,s') \big] \} \tag{1}$$

where γ denotes the discount, $A(s)$ denotes all the available actions at state s, $S(s,a)$ denotes all the available states for state s after taking action a and $R(s,a,s')$ denotes the reward when you take action a from state s and end in state s'.

- (a) the iteration results is shown in the following table

state	-2	-1	0	1	2
0 iteration	0	0	0	0	0
1 iteration	0	15.0	-5.0	26.5	0
2 iteration	0	14.0	13.45	23.0	0

- (b) After applying eq.(1) once more when the values converge and we find the optimal policy is:

state	-2	-1	0	1	2
converged	0	19.14	20.69	40.98	0
policy	0	-1	+1	+1	0

Problem 2: Transforming MDPs

- (a) Here the counterMDP is simple with start state 0 and end state 1 and -1, with rewards 100 and -100 respectively. The actions available for 0 is $+1$ and -1. The transfer probabilities are $T(0,+1,1) = 0.2, T(0,+1,-1) = 0.8$ and $T(0,-1,-1) = 0.6, T(0,-1,1) = 0.4$. And the final value of state 0 is $V_1(0) = -20$.
  With the modified transition function, we have $T'(0,+1,1) = 0.35, T'(0,+1,-1) = 0.65$ and $T'(0,-1,-1) = 0.55, T'(0,-1,1) = 0.45$. And the final value of state 0 is $V_2(0) = -10$ and thus
  $$V_2(0) > V_1(0)$$
- (b) Suppose we have an acyclic MDP. And first, we should postulate that this MDP owns finite available states, like a maze with finite size. Then, according to the definition of

'acyclic', we could claim that, taking the start state as the root node, we can derive a tree with finite depth, of which the nodes are states and the branches are actions. Consequently, a 'parent-branch-child' in the tree is related to a certain triple (s, a, s').

With all these above, we could apply an algorithm recursively by

1. get all the leaf nodes' value
2. 'delete' all the leaf nodes
3. repeat until reach the root node

This algorithm works because when deriving a leaf node's value, all required information is available and fixed.

- (c) New transition probabilities and rewards are defined as follows,

$$\begin{aligned}
T'(s,a,o) &= P(o|s,a) := 1 - P(\bar{o}|s,a) \\
T'(s,a,s') &= T(s,a,s') * P(\bar{o}|s,a) = T(s,a,s') * (1 - T'(s,a,o)) \\
Reward'(s,a,s') &= \frac{Reward(s,a,s')}{P(\bar{o}|s,a)} \\
Reward'(s,a,o) &= 0 \\
Reward'(o,a,s) &= 0
\end{aligned}$$

where s and s' do not include new state o.

The new MDP with new state o requires that every action at any state except o will have a certain probability to end in state o, which is $P(o|s,a)$. And if we set $P(\bar{o}|s,a) = \gamma \le 1$, which is the probability of not visiting state o, the new MDP(also with new Rewards) is just what we want. Here is a brief proof:

$$\begin{aligned}
U'(s) &= max_{a \in A(s)} \{ \sum_{s' \in S(s,a)} T'(s,a,s') \big[1 * U(s') + Reward'(s,a,s') \big] \} \\
&= max_{a \in A(s)} \{ \sum_{s' \in S(s,a)} T(s,a,s') * (1 - T'(s,a,o)) \big[U(s') + Reward'(s,a,s') \big] \} \\
&= max_{a \in A(s)} \{ \sum_{s' \in S(s,a)} T(s,a,s') * P(\bar{o}|s,a) \big[U(s') + \frac{Reward(s,a,s')}{P(\bar{o}|s,a)} \big] \} \\
&= max_{a \in A(s)} \{ \sum_{s' \in S(s,a)} T(s,a,s') \big[\gamma * U(s') + Reward(s,a,s') \big] \} \qquad (2)
\end{aligned}$$

Here, eq.(2) is exactly the same as eq.(1), which is what we want.

Problem 4: Learning to Play Blackjack

- (b) smallMDP

After running the simulation and the value iteration, we found that the resulting policy of the two methods are quite similar. So here, we just list those different policies. (the exploration rate is 0.2 during simulation)

state	(5, 1, (2, 1))	(5, 0, (2, 1))
value iteration policy	Take	Take
RL policy	Quit	Quit

- (b) largeMDP

After running the simulation and the value iteration, we found that the resulting policy of the two methods show many differences(about 896 different policies). So here, we just list **some of** the different policies. (the exploration rate is 0.2 during simulation)

The problem is obvious that the RL policy always QUIT when the game just starts (the total number within hand is small and safe), while it always TAKE when it is very dangerous to do so (the total number within hand is large and close to the threshold).

state	value iteration policy	RL policy
(35, None, (2, 2, 0, 1, 3))	Quit	Take
(33, 4, (2, 0, 0, 2, 3))	Quit	Take
(38, 4, (2, 2, 1, 0, 3))	Quit	Take
(9, 3, (2, 3, 3, 2, 3))	Take	Quit
(1, 3, (2, 3, 3, 3, 3))	Take	Quit
(6, 2, (2, 3, 2, 3, 3))	Take	Quit

Algorithm	sum of rewards (30,000 trials)
Fixed	205245
Q-learning(exploration rate = 0.2)	281549
Q-learning(exploration rate = 0)	360000

- (d) Here we also do 30,000 trials and we summarize all the rewards.

  From the table above we could see that Q-learning algorithm, whether with exploration or not, gains much more rewards than the fixed algorithm learnt by value iteration. The reason maybe that the environment, newThresholdMDP, is different from the originalMDP and the Q-learning algorithm performs well because of its 'learning' ability. However, the fixed algorithm learnt by value iteration on originalMDP seems to overfit the originalMDP and performs not so well when the environment is different.

  So, in summary, reinforcement learning algorithm is good at handling unknown environment and transferring to new environments, where value iteration is helpless.