Assignment 4 Blackjack

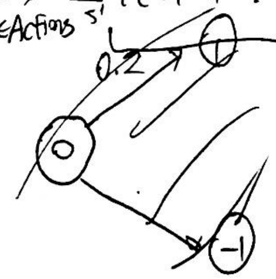
Problem I.

1.a.

· Iteration:

state	0	1	2	- - -
-2	0			
-1	0			
0	0			
1	0			
2	0			

$$V_{opt}^{(t)}(s) = \max_{a \in Actions} \underbrace{\sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_{opt}^{t-1}(s')\right]}_{Q_{opt}(s,a)}$$



$\Rightarrow$ Iteration 0: every $V_{opt}(s) = 0$.

$\Rightarrow$ Iteration 1:

State 0: $\emptyset$

~~$s'=1, a=1$.~~

$Q_{opt}(0,-1) = 0.8\times(-5+1\times 0) + 0.2\times(-5\times 1\times 0) = -5$

~~State 0:~~

$Q_{opt}(0,1) = 0.7\times(-5) + 0.3\times(-5) = -5$

$\Rightarrow V_{opt}(0) = -5$

State 1:

$Q_{opt}(1,1) = 0.7\times(-5) + 0.3\times 100 = 26.5$

$Q_{opt}(1,-1) = 0.8\times(-5) + 0.2\times 100 = 16$

$\Rightarrow V_{opt}(1) = 26.5$

State -1:

$Q_{opt}(1,-1) = 0.2\times(-5) + 20\times 0.8 = 15$

$Q_{opt}(1,1) = 0.3\times(-5) + 20\times 0.7 = 12.5$

$\Rightarrow V_{opt}(1) = 15$

$$V'_{opt}: \begin{array}{ccccc} [0, & 15, & \overset{-5}{\cancel{0}}, & 26.5, & 0] \\ -2 & -1 & 0 & 1 & -1 \end{array}$$

~~State 2:~~

~~$Q_{opt}(0,1)$~~ $=$

Iteration 2:

State 0:

$Q_{opt}(0,1) = 0.3(\cancel{\times 26.5}\,-5+26.5) + 0.7(-5+15) = 13.45$

$Q_{opt}(0,-1) = 0.2(-5+26.5) + 0.8(-5+15) = 12.3$

$\Rightarrow V_{opt}(0) = 13.45$

State 1:

$Q_{opt}(1,1) = 0.3 \times (100+0) + 0.7 \times (-5-5) = 23$

$Q_{opt}(1,-1) = 0.2(100) + 0.8(-10) = 12$

$\Rightarrow V_{opt}(1) = 23$

State -1:

$Q_{opt}(-1,1) = 0.3(-5-5) + 0.7(20) = 11$

$Q_{opt}(-1,-1) = 0.2(-5-5) + 0.8(20) = 14$

$V_{opt}(-1) = 14$

$\Rightarrow V^2_{opt}(s) = [0, 14, 13.45, 23, 0]$

$s = \; -2 \quad -1 \quad 0 \quad 1 \quad 2$

1.b. Based on $V^2_{opt}(s)$ and $Q^2_{opt}(s,a)$, π^2_{opt} is:

[N/A, -1, 1, 1, N/A]

state: -2 -1 0 1 2

Problem2.

2.b Since we know that the problem is acyclic, we can compute Vopt by following the reverse topological order using recursion.

Set $V(s_{end}) = 0$, then:

psendo code:

def findV(State):
    if state.isEnd():
        return 0
    ~~else~~
    ~~next-states = Succ(state)~~
    ~~for next~~ in next-states:
        ~~V(state) = max~~ $\cancel{\sum_{s'} T(s,a,s')[Reward(s,a,s')+\gamma V_{opt}(s')]}$ findV(next)

actions = actions(state)

recursive call.

$$V(state) = \max_{a \in actions(s)} \sum_{s'} T(s,a,s')\left(Reward(s,a,s') + \gamma \, findV(s')\right).$$

So we just need to ~~initialize~~ do one pass.

2C. $Q_{opt}^{(t+1)}(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{opt}^{(t+1)}(s')]$

If we define $T'(s,a,s') = \gamma T(s,a,s')$

and $T'(s,a,o) = (1-\gamma)\, \cancel{T(s,a}\, \cancel{\Xi}(\sum_{s'} T(s,a,s'))$

$\Rightarrow$

$Q_{opt}'^{(t-1)}(s,a) = \sum_{s' \neq o} T(s,a,s')\gamma[\text{Reward}'(s,a,s') + V_{opt}'^{(t-1)}(s')]$

since discount $= 1$ in this case

$\Rightarrow Q_{opt}'^{(t+1)}(s,a) = \sum_{s'} [T(s,a,s')\gamma \text{Reward}'(s,a,s') + T(s,a,s')\gamma V_{opt}'^{(t-1)}(s')]$

$+ T'(s,a,o)\text{Reward}'(s,a,o) + T'(s,a,o) V_{opt}^{(t-1)}(o)$

$\Rightarrow$ We want $\cancel{Op}\; Q_{opt}'^{(t+1)}(s,a) = Q_{opt}^{(t+1)}(s,a)$ — ①

$\Rightarrow Q_{opt}^{(t+1)}(s,a) = T(s,a,s')R(s,a,s') + T(s,a,s')\gamma \cancel{\theta} V_{opt}^{(t+1)}(s')$ — ②

$\Rightarrow$ Compare ① and ②:

$\text{Reward}'(s,a,s') = \dfrac{\text{Reward}(s,a,s')}{\gamma}$

$\text{Reward}'(s,a,o) = 0$

$T'(s,a,s') = \gamma T(s,a,s')$; $T'(s,a,o) = (1-\gamma)\sum_{s'} T(s,a,s')$

$T'(o,a,s') = 0$ therefore $V_{opt}(o) = 0$.

$R'(o,a,s') = 0$

Problem 4.

b. After running smallMDP and largeMDP using both Q-learning and valueIteration, it turned out that the discrepancy was larger on largeMDP than on smallMDP. This was due to the fact that largeMDP has more possible states to explore. Also, the feature extractor was not very good because it extracted specific state-action pairs which are hard to be applied on future datasets. (In 4c words, poor generalization)

d. We got average reward = 6.2 from applying the policy got from originalMDP onto newThresholdMDP. We got average reward = 12.0 when running Q-learning on newThresholdMDP directly. The reson for the difference is that

Q-learning is able to find a new optimal policy when presented a different senario, while FixedRL-Algorithm ~~could~~ can not.