

Homework I

Deadline: 2022-10-19

1. (5 pts) Show that the infinite horizon discounted state value $v_\pi(s)$ has the following alternative expression:

$$v_\pi(s) = \mathbb{E}_{N \sim \text{Geo}(1-\gamma)} \left[\mathbb{E} \left[\sum_{t=0}^{N-1} r(s_t, a_t, s_{t+1}) | s_0 = s \right] \right],$$

where $\text{Geo}(1-\gamma)$ denotes the geometric distribution with parameter $1-\gamma$. In word, we can rewrite $v_\pi(s)$ into an undiscounted form where the length of trajectory obeys the geometric distribution. In addition, compute $\mathbb{E}[N]$ which is referred to as planning horizon.

2. (10 pts) Prove the Bellman equation for state value and action value functions (i.e., Theorem 1 in the latest version of Lecture1.pdf).
3. (5 pts) Show that the Bellman operator

$$\mathcal{T}_\pi v = r_\pi + \gamma P^\pi v$$

is a contraction with respect to infinity norm.

4. (5 pts) Whether the optimal policy is unique? Prove or disprove by a counter example.
5. (optional) Let π_k be the policy extracted from the k -th iteration of the value iteration. Is it always that $v_{\pi_{k+1}}(s) \geq v_{\pi_k}(s)$, $\forall s$? Prove or disprove by a counter example.
6. (5 pts) Prove the policy improvement result (i.e., Theorem 2 in the latest version of Lecture2.pdf).
7. (20 pts)

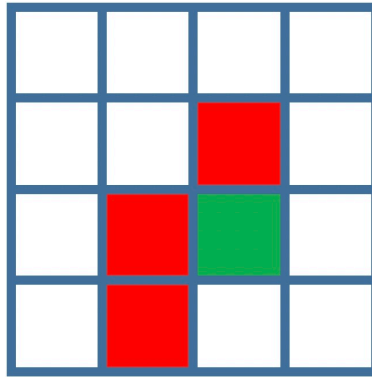


Figure 1: GridWorld Example

Consider the gridworld problem shown in Fig. 1. Here are the basic settings:

- There are sixteen states with three obstacles (red) and one target (green).
- At each state, there are five available actions (up, down, left, right, stay). For those states on the boundaries, if taking an action causes the agent to leave the grids, the agent will return back. For the goal state, no matter what actions are taken, the agent will always return back.
- The reward is -1 if the agent enters the obstacle grid, is 1 if the agent enters the goal grid, is 0 for other grids.

Implement value iteration and policy iteration to find the optimal policy for this problem for two different discount factors $\gamma = 0.9$ and $\gamma = 0.5$. Are the optimal policies for those two cases the same? How can you interpret your observation?

The sample codes are provided (only) for the first time for asynchronous value iteration, see them attached (correctness is not guaranteed). Indeed, you can find more sample codes in <https://github.com/boyu-ai/Hands-on-RL> However, simple copy & paste will not be helpful at all.

```
1 import numpy as np
2
3 # Create environment
4 class GridWorld:
5     def __init__(self,nrow=4, ncol=4):
6         self.nrow = nrow
7         self.ncol = ncol
8         self.P = self.createP() # P[s][a] = [(s',r)]
9
10    def createP(self):
11        acts = [[-1, 0], [1, 0], [0, -1], [0, 1], [0, 0]]
12
13        # the inside [] means the entry of P is an array
14        P = [[ [] for j in range(5)] for i in range(self.nrow*self.ncol)]
15
16        # five actions
17        # acts[0]: up, acts[1]: down, acts[2]: left, acts[3]: right, acts[4
18    ]: stay
19
20    for i in range(self.nrow):
21        for j in range(self.ncol):
22            s=[i,j]
23            #print(s)
24            #print('=====' )
25            for a in range(5):
26                sp = np.add(s,acts[a])
27                if sp[0]==-1 or sp[0] == self.nrow or sp[1]==-1 or sp[1]
```

```

26 ]=self.ncol or (i==2 and j==2): # boundary and goal
27     sp[0]=i
28     sp[1]=j
29
30     if i==2 and j==2: # target
31         #print(sp)
32         #print('-----')
33         P[i*self.ncol+j][a] = [sp[0]*self.ncol+sp[1],1]
34     elif (i==1 and j==2) or (i==2 and j == 1) or (i==3 and j
    =1): # obstacles
35         P[i*self.ncol+j][a] = [sp[0]*self.ncol+sp[1],-1]
36     else: # other states
37         #print(sp)
38         #print('-----')
39         P[i*self.ncol+j][a] = [sp[0]*self.ncol+sp[1],0]
40     return P
41
42 # Implement asynchronous VI
43 class ValueIteration:
44     def __init__(self, env, gamma, eps):
45         self.env = env
46         self.v = [0] * self.env.nrow * self.env.ncol
47         self.gamma = gamma
48         self.eps = eps
49         self.pi = [None for i in range(self.env.nrow * self.env.ncol)]
50
51     def value_iteration(self):

```

```

52         iter = 0
53         err_inf = float('inf')
54         while 1:
55             if err_inf < self.eps:
56                 break
57
58             err_inf = 0
59             for s in range(self.env.nrow * self.env.ncol):
60                 new_vs = float('-inf')
61                 for a in range(5):
62                     # print(self.env.P[s][a])
63                     sp, r = self.env.P[s][a]
64                     qsa = r + self.gamma * self.v[sp]
65                     if qsa ≥ new_vs:
66                         new_vs = qsa
67                         self.pi[s] = a
68                     # new_vs = max(new_vs, qsa)
69
70             err_inf = max(err_inf, abs(new_vs - self.v[s]))
71             self.v[s] = new_vs
72
73             iter += 1
74             print('Iter: %d' % iter, 'Error: %.4f' % err_inf)
75
76     def print_pi(self):
77         action = ['^', 'v', '<', '>', 'o']
78         policy = np.empty((self.env.nrow, self.env.ncol), dtype=object)

```

File - /Users/kewei/Teaching/Codes/hw1.py

```
79         for i in range(self.env.nrow):
80             for j in range(self.env.ncol):
81                 a = self.pi[i * self.env.ncol + j]
82                 policy[i, j] = action[a]
83         print(policy)
84
85 # Begin test ...
86 env = GridWorld()
87 eps = 0.0001
88 gamma = 0.9
89 agent = ValueIteration(env, gamma, eps)
90
91 agent.value_iteration()
92 agent.print_pi()
```

Page 4 of 4