Homework I

Deadline: 2024-10-14

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

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

   where $Geo(1-\gamma)$ 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. (5 pts) Whether the optimal policy is unique? Prove or disprove by a counter example.

3. (5 pts) Given any vector $V \in \mathbb{R}^{|\mathcal{S}|}$, let π be the greedy policy defined from V. Is it true $V^\pi = V$?

4. (5 pts) 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.

5. (10 pts) Write out the value iteration in terms of action values based on the related Bellman optimality condition and show the linear convergence of the algorithm.

6. (10 pts) Given a policy π, define the advantage value as follows:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s).$$

   Show that if $A^\pi(s,a) \leq 0$, then π is an optimal policy.

7. (10 pts) Reproduce the figure on pg. 22 of Lecture 3 for the test of RM on root finding.

8. (20 pts)

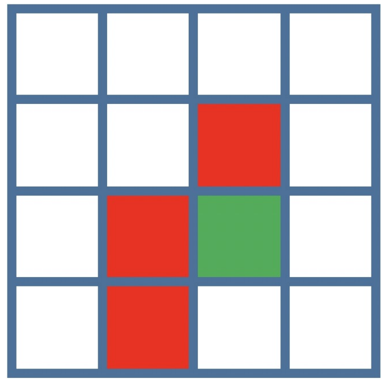

Figure 1: GridWorld Example

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.

```
import numpy as np

# Create environment
class GridWorld:
    def __init__(self,nrow=4, ncol=4):
        self.nrow = nrow
        self.ncol = ncol
        self.P = self.createP() # P[s][a] = [(s',r)]

    def createP(self):
        acts = [[-1, 0], [1, 0], [0, -1], [0, 1], [0, 0]]

        # the inside [] means the entry of P is an array
        P = [[ [] for j in range(5)] for i in range(self.nrow*self.ncol)]

        # five actions
        # acts[0]: up, acts[1]: down, acts[2]: left, acts[3]: right, acts[4]: stay

        for i in range(self.nrow):
            for j in range(self.ncol):
                s=[i,j]
                #print(s)
                #print('=======')
                for a in range(5):
                    sp = np.add(s,acts[a])
                    if sp[0]==-1 or sp[0] == self.nrow or sp[1]==-1 or sp[1
```

```
]==self.ncol or (i==2 and j==2): # boundary and goal
                        sp[0]=i
                        sp[1]=j

                    if i==2 and j==2: # target
                        #print(sp)
                        #print('-----')
                        P[i*self.ncol+j][a] = [sp[0]*self.ncol+sp[1],1]
                    elif (i==1 and j==2) or (i==2 and j == 1) or (i==3 and j==1): # obstacles
                        P[i*self.ncol+j][a] = [sp[0]*self.ncol+sp[1],-1]
                    else: # other states
                        #print(sp)
                        #print('-----')
                        P[i*self.ncol+j][a] = [sp[0]*self.ncol+sp[1],0]
        return P

# Implement asynchronous VI
class ValueIteration:
    def __init__(self, env, gamma, eps):
        self.env = env
        self.v = [0] * self.env.nrow * self.env.ncol
        self.gamma = gamma
        self.eps = eps
        self.pi = [None for i in range(self.env.nrow * self.env.ncol)]

    def value_iteration(self):
```

```python
        iter = 0
        err_inf = float('inf')
        while 1:
            if err_inf < self.eps:
                break

            err_inf = 0
            for s in range(self.env.nrow * self.env.ncol):
                new_vs = float('-inf')
                for a in range(5):
                    # print(self.env.P[s][a])
                    sp, r = self.env.P[s][a]
                    qsa = r + self.gamma * self.v[sp]
                    if qsa >= new_vs:
                        new_vs = qsa
                        self.pi[s] = a
                    # new_vs = max(new_vs,qsa)

                err_inf = max(err_inf, abs(new_vs - self.v[s]))
                self.v[s] = new_vs

            iter += 1
            print('Iter: %d' % iter, 'Error: %.4f' % err_inf)

    def print_pi(self):
        action = ['^', 'v', '<', '>', 'o']
        policy = np.empty((self.env.nrow, self.env.ncol), dtype=object)
```

```
        for i in range(self.env.nrow):
            for j in range(self.env.ncol):
                a = self.pi[i * self.env.ncol + j]
                policy[i, j] = action[a]
        print(policy)

# Begin test ...
env = GridWorld()
eps = 0.0001
gamma = 0.9
agent = ValueIteration(env, gamma, eps)

agent.value_iteration()
agent.print_pi()
```