Assignment 2

Peng Tang 517020910038

In this assignment, we implement first-visit and every-visit MC method and TD(0) to evaluate the random policy on the given small gridworld.

1 Analysis

1.1 First-visit MC

In order to reduce memory use, we use increment Monte-Carlo to update the value of every states. We implement first-visit MC as following steps.

- Producing a episode following current policy π.

  Because current policy π is random policy, we randomly choose a state as the start state, then randomly take actions at each state until reaching the terminal states. We record all states we reach in the process.

- After gaining a episode, we traverse the episode and update the value of corresponding states by using the following equations. And for first-visit MC, we just update a state when we visit it for the first time in the episode.

$$G(s_t) = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1-t} R_n = \begin{cases} (-1) \times (n-t) & \gamma = 1 \\ (-1) \times \frac{(1-\gamma^{n-t})}{1-\gamma} & 0 < \gamma < 1 \end{cases}$$

$$N(s_t) = N(s_t) + 1$$

$$V(s_t) = V(s_t) + \frac{1}{N(s_t)}(G(s_t) - V(s_t))$$

Where s_t means the t-th state in the episode and n means the number of states in the episode. And for this problem, we have $R_{t+1} = \cdots = R_n = -1$. γ is the discount factor.

We do above two steps for enough times, then the values of all states will converge to fixed values.

1.2 Every-visit MC

The difference between every-visit MC and first-visit MC is that every-visit MC need update a state every time we visit it. So we can implement this just like first-visit MC.

1.3 TD(0)

For TD(0), we update the value of states for every step rather than producing a episode. We can do this as following steps.

- We randomly choose a start state and take random actions until reaching a terminal state.
- In the process, everytime we take an action, we can update the value of current state by using following equations.

$$V(s) = V(s) + \alpha(G - V(s)) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

Where s' is the next state of s after taking an action and $r = -1$.

- We do above two steps for enough times and the values of all states will converge to fixed values.

2 Code

In the *solution.py*, we implement MC method and TD(0) method by defining two class, **MC** and **TD**.

2.1 MC Class

```
class MC(object):

    def __init__(self, discount):

        self.FirstValue =  np.random.randn(6,6)  #初始化所有位置的值
        self.EveryValue =  np.random.randn(6,6) #初始化所有位置的值
        self.FirstValue[0][1] = 0
        self.FirstValue[5][5] = 0
        self.EveryValue[0][1] = 0
        self.EveryValue[5][5] = 0


        self.reward = -1
        self.discount = discount
        self.action = [[-1,0],[0,1],[1,0],[0,-1]]
        self.terminalState = np.array([[0,1],[5,5]])


    def produce_random_episodes(self):
        episodesList = []

        state = np.random.randint(0,6,2)
        episodesList.append(state.copy())
        while (state!=self.terminalState[0]).any() and (state!=self.terminalState[1]).any():
            a = np.random.randint(0,4,1)
            state[0] = max(0,min(5,state[0]+self.action[a[0]][0]))
            state[1] = max(0,min(5,state[1]+self.action[a[0]][1]))

            episodesList.append(state.copy())

        return episodesList

    def first_visit_MC(self,maxEpisodes):
        count = np.zeros((6,6))

        times = 1
        error = []
        while times < maxEpisodes:

            tempcount = np.zeros((6,6))
            episd = self.produce_random_episodes()

            n = len(episd)
```

```python

            maxerror = 0
            for i in range(n-1):

                if tempcount[episd[i][0]][episd[i][1]] == 0:
                    tempcount[episd[i][0]][episd[i][1]] = 1
                    count[episd[i][0]][episd[i][1]] +=1
                    Gt = 0
                    if self.discount == 1:
                        Gt = self.reward*(n-i-1)
                    else:
                        Gt = self.reward*((1-self.discount**(n-i-1))/(1-self.discount))

                    oldval = self.FirstValue[episd[i][0]][episd[i][1]]
                    self.FirstValue[episd[i][0]][episd[i][1]] += 1./count[episd[i][0]][episd[i][1]]*(Gt-self.FirstValue[episd[i][0]][episd[i][1]])

                    maxerror = max(maxerror,abs(oldval-self.FirstValue[episd[i][0]][episd[i][1]]))
            times+=1
            error.append(maxerror)

        return error

    def every_visit_MC(self,maxEpisodes):

        count = np.zeros((6,6))

        times = 1
        error = []
        while times < maxEpisodes:

            episd = self.produce_random_episodes()
            n = len(episd)
            maxerror = 0
            for i in range(n-1):
                    count[episd[i][0]][episd[i][1]] +=1
                    Gt = 0
                    if self.discount == 1:
                        Gt = self.reward*(n-i-1)
                    else:
                        Gt = self.reward*((1-self.discount**(n-i-1))/(1-self.discount))
                    oldval = self.EveryValue[episd[i][0]][episd[i][1]]
                    self.EveryValue[episd[i][0]][episd[i][1]] += 1./count[episd[i][0]][episd[i][1]]*(Gt-self.EveryValue[episd[i][0]][episd[i][1]])
                    maxerror = max(maxerror,abs(oldval-self.EveryValue[episd[i][0]][episd[i][1]]))
            times+=1
            error.append(maxerror)
        return error
```

- In the class, **FirstValue** is used to store the values for first-visit method; **EveryValue** is used to store the values for every-visit method. **reward** is always -1 in this problem; **discount** is the discount factor; **action** contains four actions that we can take; **terminalState** contains two terminal states.

- The **produce_random_episodes()** function is used to produce a episode randomly and return a list containing all states of the episode.
- The **first_visit_MC()** function is used to compute the values with first-visit MC method. In this function, we need a parameter as the terminal condition. We implement the function as the analysis.

- The **every_visit_MC()** function is used to compute the values with every-visit MC method. In this function, we need a parameter as the terminal condition. We implement the function as the analysis.

2.2 TD Class

```
class TD(object):
    def __init__(self,discount,stepSize):

        self.Value = np.random.randn(6,6)

        self.Value[0][1] = 0
        self.Value[5][5] = 0
        self.reward = -1
        self.discount = discount
        self.stepSize = stepSize
        self.action = [[-1,0],[0,1],[1,0],[0,-1]]
        self.terminalState = np.array([[0,1],[5,5]])


    def TD_find(self, maxepisods):

        times = 1
        error = []
        while times<maxepisods:
            self.stepSize = self.stepSize*0.99
            maxerror = 0
            state = np.random.randint(0,6,2)
            while (state!=self.terminalState[0]).any() and 
(state!=self.terminalState[1]).any():
                a = np.random.randint(0,4,1)

                oldval = self.Value[state[0]][state[1]]
                oldstate = state.copy()
                state[0] = max(0,min(5,state[0]+self.action[a[0]][0]))
                state[1] = max(0,min(5,state[1]+self.action[a[0]][1]))
                self.Value[oldstate[0]][oldstate[1]] = self.stepSize*
(self.reward + self.discount*self.Value[state[0]][state[1]]) + (1-
self.stepSize)*self.Value[oldstate[0]][oldstate[1]]

                maxerror = max(maxerror, abs(oldval-self.Value[oldstate[0]]
[oldstate[1]]))
            times+=1

            error.append(maxerror)
        return error
```

- In this class, the **stepSize** is the weight factor α we use to update the value.

- The **TD_find()** function is used to compute the values with TD(0) method. In this function, we need a parameter as the terminal condition. We implement the function as the analysis.

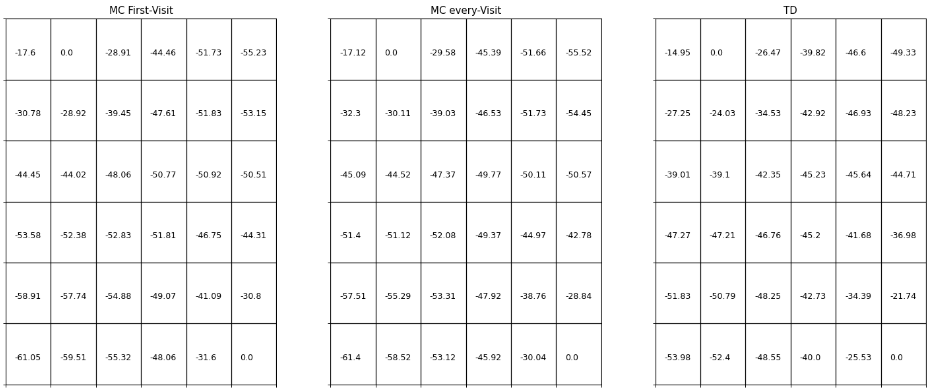
2.3 Main function

```
def main():
    MC_discount = 1
    MC_maxEpsides = 5000
    MC_model = MC(MC_discount)
    t1 = time.time()
    Error1 = MC_model.first_visit_MC(MC_maxEpsides)
    t2 = time.time()
    Error2 = MC_model.every_visit_MC(MC_maxEpsides)
    t3 = time.time()

    print("The spent time of MC first visit: %s" %(t2-t1))
    print("The spent time of MC every visit: %s" %(t3-t2))



    #TD
    TD_discount = 1
    TD_maxEpsides = 5000
    TD_stepsize = 0.8
    TD_model = TD(TD_discount,TD_stepsize)
    t4 = time.time()
    Error3 = TD_model.TD_find(TD_maxEpsides)
    t5 = time.time()
    print("The spent time of TD: %s" %(t5-t4))



    plotResult(MC_model.FirstValue,MC_model.EveryValue,TD_model.Value)

    poltError(Error1,Error2,Error3)
```

This function is used to call the methods we implement and produce the results.

3 Results

By running the code, we can get following results.



MC First-Visit

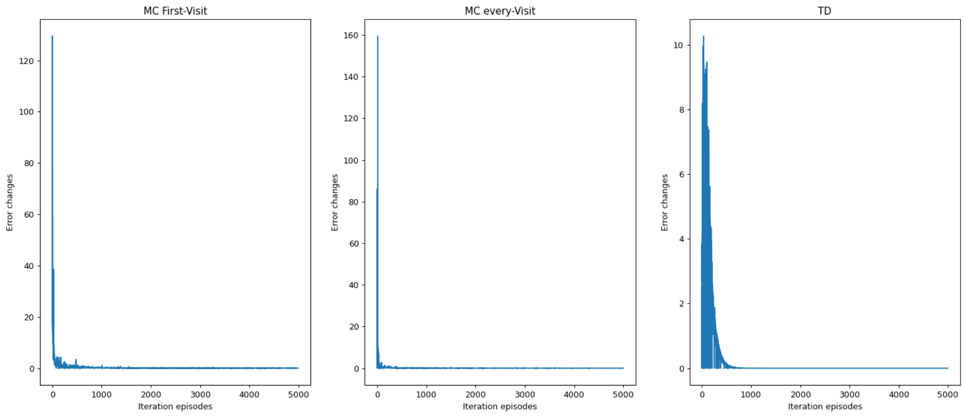
-17.6	0.0	-28.91	-44.46	-51.73	-55.23
-30.78	-28.92	-39.45	-47.61	-51.83	-53.15
-44.45	-44.02	-48.06	-50.77	-50.92	-50.51
-53.58	-52.38	-52.83	-51.81	-46.75	-44.31
-58.91	-57.74	-54.88	-49.07	-41.09	-30.8
-61.05	-59.51	-55.32	-48.06	-31.6	0.0

MC every-Visit

-17.12	0.0	-29.58	-45.39	-51.66	-55.52
-32.3	-30.11	-39.03	-46.53	-51.73	-54.45
-45.09	-44.52	-47.37	-49.77	-50.11	-50.57
-51.4	-51.12	-52.08	-49.37	-44.97	-42.78
-57.51	-55.29	-53.31	-47.92	-38.76	-28.84
-61.4	-58.52	-53.12	-45.92	-30.04	0.0

TD

-14.95	0.0	-26.47	-39.82	-46.6	-49.33
-27.25	-24.03	-34.53	-42.92	-46.93	-48.23
-39.01	-39.1	-42.35	-45.23	-45.64	-44.71
-47.27	-47.21	-46.76	-45.2	-41.68	-36.98
-51.83	-50.79	-48.25	-42.73	-34.39	-21.74
-53.98	-52.4	-48.55	-40.0	-25.53	0.0




```
PS E:\大三下\强化学习\作业\homework2\code> D:\python\python.exe "e:\大三下\强化学习\作业\homework2\code\solution.py"
The spent time of MC first visit: 4.1528167724609375
The spent time of MC every visit: 4.19720721244812
The spent time of TD: 4.327538967132568
```

- From the first figure, we can see that first-visit MC and every-visit MC give similar results, and the result of TD(0) is lightly different from their.
- From the second figure, we can see that the convergence rate of every-visit MC and TD(0) is faster than first-visit MC. And the change of TD's error is smaller.
- From the second figure, we can see that the running time of three methods is almost same.