日期	科目	班级	姓名	学号
2023 年 4 月 7 日	强化学习	强基数学 002	吴天阳	2204210460

第四次作业　第七章

题目 1. 练习 7.2 在 n 步方法中，价值函数需要每步都更新，所以利用 TD 误差值和代替下述公式

$$V(S_t) \leftarrow V(S_t) + \alpha[G_{t:t+n} - V(S_t)], \quad 0 \leqslant t < T$$

中的错误项的算法将会与之前不同. 这种算法是一个更好的还是更差的算法？请设计一个小实验并编程验证这个问题.

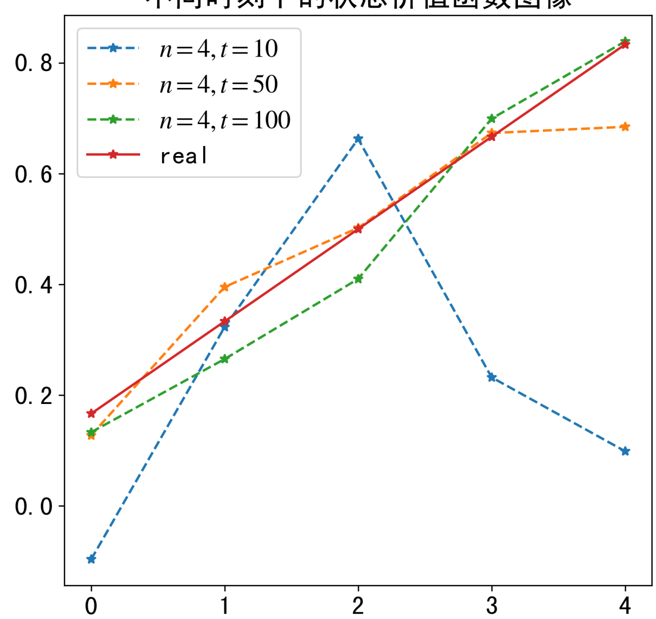
解答. 类似 MC 算法，如果不考虑每步都进行更新，则也可表示为 TD 误差值和；在 n 步 TD 方法中，如果不考虑每步都更新，则可写为 n 个 TD 误差和

$$\begin{aligned}
G_{t:t+n} - V(S_t) = &R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\\
&+\gamma(R_{t+2} + \cdots + \gamma^{n-2}R_{t+n} + \gamma^{n-1}V(S_{t+n}) - V(S_{t+1}))\\
=&\delta_t + \gamma(R_{t+2} + \gamma V(S_{t+2}) - V(S_{t+1}))\\
&+\gamma^2(R_{t+3} + \cdots \gamma^{n-3}R_{t+n} + \gamma^{n-2}V(S_{t+n}) - V(S_{t+2}))\\
=&\delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{n-2}\delta^{t+n-2}\\
&+\gamma^{n-1}(R_{t+n} + \gamma V(S_{t+n}) - V(S_{t+n-1})) + \gamma^n(V(S_{t+n}) - V(S_{t+n}))\\
=&\delta_t + \gamma\delta_{t+1} + \cdots \gamma^{n-1}\delta_{t+n-1} = \sum_{k=t}^{t+n-1}\gamma^{k-t}\delta_k
\end{aligned}$$

所以该题的问题就是比较 n 步 TD 算法的实时更新和非实时更新（每次一幕结束之后，更新状态价值函数）的好坏.

我使用的例子是书上**例 6.2 随机游走**：A~E 的真实价值分别为 $\{\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}\}$，使用实时更新 4 步 TD 算法状态价值函数（Real time n-TD）图像如右图所示.

不同时刻下的状态价值函数图像



我绘制了实时 TD 算法（Real time）和非实时 TD 算法（non-Real time）的均方误差和幕数的变换关系图，容易看出，实时 TD 算法的均方误差一直比非实时 TD 算法要小，说明实时 TD 算法更优，因为在实时更新算法中智能体学习过程中，仍在不断更新状态价值函数，从而可以更快地收敛到真实状态价值函数.

但二者差别并不是很大，因为策略是固定的，如果在策略迭代中，实时算法收敛速度应该更加迅速.

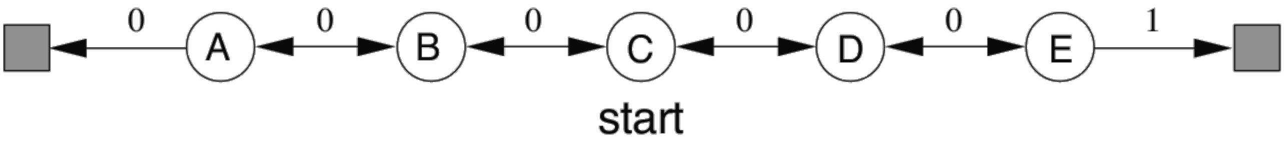
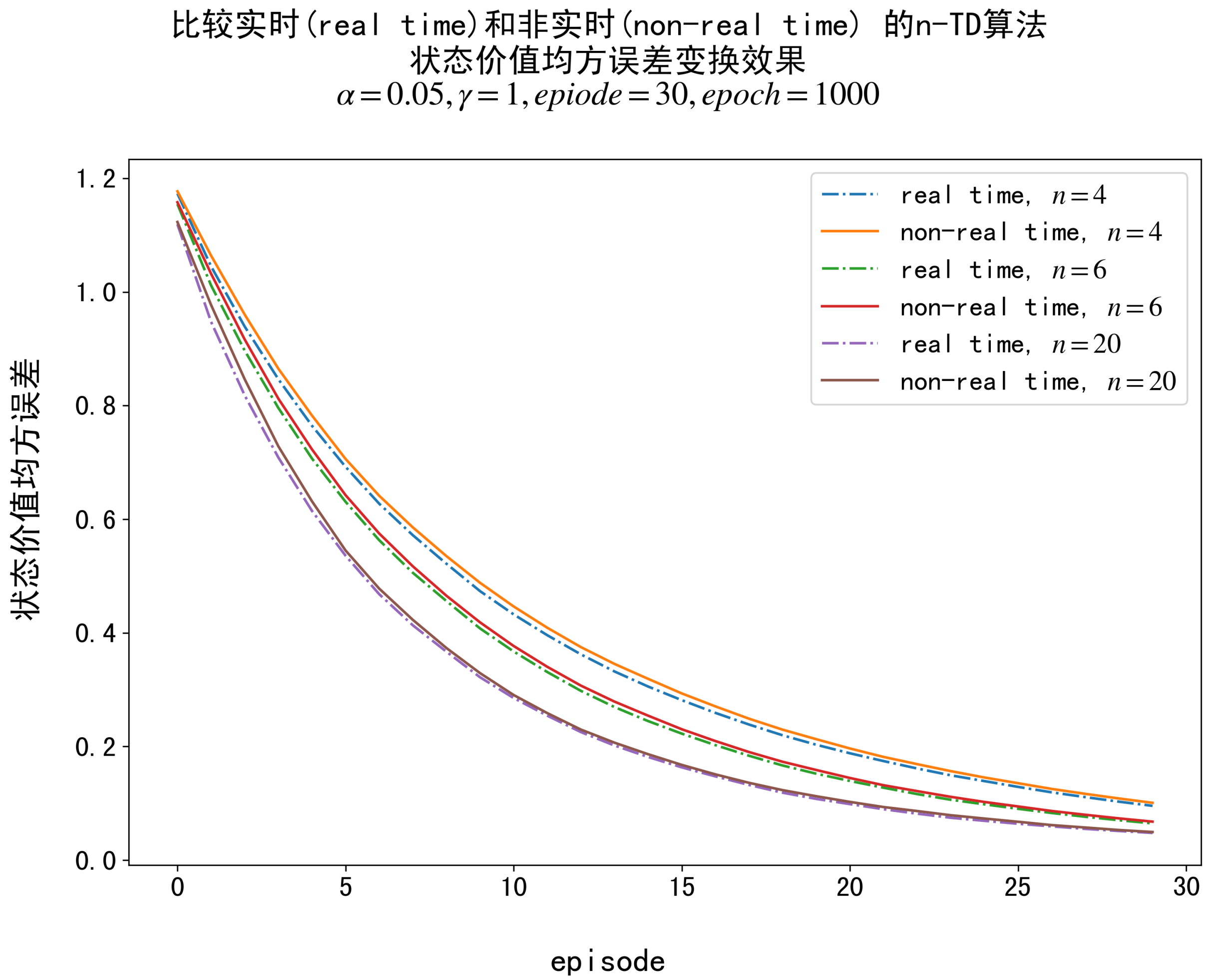

0
A
0
B
0
C
0
D
0
E
1
start

图 1: 随机游走


比较实时(real time)和非实时(non-real time) 的n-TD算法
状态价值均方误差变换效果
$\alpha=0.05, \gamma=1, epiode=30, epoch=1000$
状态价值均方误差
episode
real time, $n=4$
non-real time, $n=4$
real time, $n=6$
non-real time, $n=6$
real time, $n=20$
non-real time, $n=20$
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

完整代码：

```python
# -*- coding: utf-8 -*-
'''
@File    : main.py
@Time    : 2023/03/21 19:43:45
@Author  : wty-yy
@Version : 1.0
@Blog    : https://wty-yy.space/
@Desc    : 《强化学习》中文书第 141 页练习 7.2，比较 n-TD 算法的两种实现方法
        1. 实时更新与书上写法一致。
        2. 进行完一回合之后更新一次。
        两种算法均对状态价值函数进行估计，选的例子是第 123 页的“随机游走”，
        随机游走例子中每个节点的真实状态价值函数都是已知的，
        通过绘制“步数-价值函数的均方误差”的图像来比较两种算法。
'''

import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from template.reinforcement import Environment, Agent
from pathlib import Path
PATH_FIGURES = Path(__file__).parent

class RandomWalk(Environment):
    avail_points = 5
    def __init__(self, avail_points=5) -> None:
        super().__init__()
        self.avail_points = avail_points
        self.rewards = [0 for _ in range(self.avail_points + 1)]
        self.rewards.append(1)
        self.true_state_value = [0, *[(i+1)/(1+self.avail_points) for i in range(self.avail_points)], 0]

    def reset(self):
        self.state = self.avail_points // 2 + 1
        return self.state

    def step(self, action):
        self.state += action
        reward = self.rewards[self.state]
        terminal = 0 if self.state != 0 and self.state != self.avail_points + 1 else 1
        return reward, self.state, terminal

class n_TD_Agent(Agent):
    def __init__(self, n_TD=6, alpha=0.05, epsilon=0, gamma=1, seed=42, episode=30, epoch=1000) -> None:
        super().__init__(alpha, epsilon, gamma, seed, episode, epoch)
        self.n_TD = n_TD

    def start(self):
        for real_time in [True, False]:
```

```
        # for real_time in [True]:
            self.history = np.zeros(self.episode)
            for _ in tqdm(range(self.epoch)):
                self.history += (self.train(real_time) - self.history) / (1 + _)
            self.plot_MSE(label=f"{'' if real_time else 'non-'}real time, $n={self.n_TD}$", ls='-.' if real_time else '-')

    def train(self, real_time=True, verbose=False, **kargs):
        def choose_action():
            return 1 if np.random.rand(1)[0] > 0.5 else -1

        history = []
        n = self.n_TD
        state_value = np.random.normal(size=RandomWalk.avail_points + 2)
        state_value[0] = state_value[-1] = 0

        for _ in range(self.episode):
            env = RandomWalk()
            state = env.reset()
            t, T = 0, np.inf
            # rewards, states = ([0 for _ in range(n + 1)] for _ in range(2))
            # states[0] = state
            rewards, states = [0], []
            states.append(state)

            def update_state_value():
                if tau >= 0:
                    target = 0
                    for i in range(tau + 1, min(tau + n, T) + 1):
                        # target += np.power(self.gamma, i - tau - 1) * rewards[i % (n+1)]
                        target += np.power(self.gamma, i - tau - 1) * rewards[i]
                    if tau + n < T:
                        # target += np.power(self.gamma, n) * state_value[states[(tau+n) % (n+1)]]
                        target += np.power(self.gamma, n) * state_value[states[tau+n]]
                    # delta = target - state_value[states[tau % (n+1)]]
                    delta = target - state_value[states[tau]]
                    # state_value[states[tau % (n+1)]] += self.alpha * delta
                    state_value[states[tau]] += self.alpha * delta

            while True:
                if t < T:
                    action = choose_action()
                    reward, state, terminal = env.step(action)
                    rewards.append(reward)
                    states.append(state)
                    # rewards[(t+1) % (n+1)] = reward
                    # states[(t+1) % (n+1)] = state
                    if terminal:
                        T = t + 1
                tau = t - n + 1
```

```
                if real_time:
                    update_state_value()
                if tau == T - 1:
                    break
                t += 1
            if not real_time:
                for tau in range(T):
                    update_state_value()

            mse = np.sum(np.power(state_value - env.true_state_value, 2)) / (env.avail_points)
            history.append(mse)
            if verbose and (_+1) in kargs['verbose_time']:
                plt.plot(state_value[1:-1], '--*', label=f"$n={n}, t={_+1}$")
        if verbose:
            plt.plot(env.true_state_value[1:-1], '-*', label="real")

        return np.array(history)

    def plot_MSE(self, label, ls):
        plt.plot(self.history, label=label, ls=ls)

if __name__ == '__main__1':
    plt.title(" 不同时刻下的状态价值函数图像")
    agent = n_TD_Agent(n_TD=4, alpha=0.05, episode=100)
    agent.train(real_time=True, verbose=True, verbose_time=[10, 50, 100])
    plt.legend()
    plt.tight_layout()
    plt.savefig(PATH_FIGURES.joinpath("state value function in diff time.png"), dpi=300)
    plt.show()

if __name__ == "__main__":
    fig = plt.figure(figsize=(10, 8))
    for subid, n in enumerate([4, 6, 20]):
        agent = n_TD_Agent(n_TD=n)
        agent.start()
        plt.legend()
    fig.suptitle(" 比较实时 (real time) 和非实时 (non-real time) 的 n-TD 算法\n状态价值均方误差变换效果\n"

                    +f"$\\alpha={agent.alpha},\\gamma={agent.gamma},epiode={agent.episode
    fig.supylabel(" 状态价值均方误差")
    fig.supxlabel("episode")

    plt.tight_layout()
    plt.savefig(PATH_FIGURES.joinpath("compare (non)real time n-TD.png"), dpi=300)
    plt.show()
```