Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Лабораторная работа №5
по дисциплине «Методы машинного обучения»
«Обучение на основе временных различий»

ИСПОЛНИТЕЛЬ:

Алехин С.С.
Группа ИУ5-23М

_____

ПРОВЕРИЛ:

Балашов А.М.

_____

Москва, 2023

Задание

На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

- SARSA
- Q-обучение
- Двойное Q-обучение

Для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).

Lab5

June 21, 2023

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import gymnasium as gym
from tqdm import tqdm
```

0.0.1

[ ]:
```
class BasicAgent:
    '''
          ,
    '''

    #
    ALGO_NAME = '---'

    def __init__(self, env, eps=0.1):
        #
        self.env = env
        #        Q-
        self.nA = env.action_space.n
        self.nS = env.observation_space.n
        #
        self.Q = np.zeros((self.nS, self.nA))
        #
        #
        self.eps=eps
        #
        self.episodes_reward = []

    def print_q(self):
        print('    Q-             ', self.ALGO_NAME)
        print(self.Q)

    def get_state(self, state):
        '''

        '''
        if type(state) is tuple:
```

```
            #                          ,
            return state[0]
        else:
            return state

    def greedy(self, state):
        '''
        <<   >>
                  ,                    Q-
              state
        '''
        return np.argmax(self.Q[state])

    def make_action(self, state):
        '''

        '''
        if np.random.uniform(0,1) < self.eps:

            #                eps
            #
            return self.env.action_space.sample()
        else:
            #          ,                    Q-
            return self.greedy(state)

    def draw_episodes_reward(self):
        #
        fig, ax = plt.subplots(figsize = (15,10))
        y = self.episodes_reward
        x = list(range(1, len(y)+1))
        plt.plot(x, y, '-', linewidth=1, color='green')
        plt.title('            ')
        plt.xlabel('        ')
        plt.ylabel('    ')
        plt.show()

    def learn():
        '''

        '''
        pass
```

0.0.2 SARSA

[ ]:
```
class SARSA_Agent(BasicAgent):
    '''
                SARSA
    '''
    #
    ALGO_NAME = 'SARSA'

    def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
        #
        super().__init__(env, eps)
        # Learning rate
        self.lr=lr
        #
        self.gamma = gamma
        #
        self.num_episodes=num_episodes
        #               eps
        self.eps_decay=0.00005
        self.eps_threshold=0.01

    def learn(self):
        '''
                        SARSA
        '''
        self.episodes_reward = []
        #
        for ep in tqdm(list(range(self.num_episodes))):
            #
            state = self.get_state(self.env.reset())
            #
            done = False
            #
            truncated = False
            #
            tot_rew = 0

            #             Q-
            if self.eps > self.eps_threshold:
                self.eps -= self.eps_decay

            #
            action = self.make_action(state)

            #
            while not (done or truncated):
                #
```

```
                next_state, rew, done, truncated, _ = self.env.step(action)

                #
                next_action = self.make_action(next_state)

                #            Q    SARSA
                self.Q[state][action] = self.Q[state][action] + self.lr * \
                    (rew + self.gamma * self.Q[next_state][next_action] - self.
↪Q[state][action])

                #
                state = next_state
                action = next_action
                #
                tot_rew += rew
                if (done or truncated):
                    self.episodes_reward.append(tot_rew)
```

0.0.3 Q-

[ ]:
```
class QLearning_Agent(BasicAgent):
    '''
                Q-Learning
    '''
    #
    ALGO_NAME = 'Q-     '

    def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
        #
        super().__init__(env, eps)
        # Learning rate
        self.lr=lr
        #
        self.gamma = gamma
        #
        self.num_episodes=num_episodes
        #               eps
        self.eps_decay=0.00005
        self.eps_threshold=0.01

    def learn(self):
        '''
                        Q-Learning
        '''
        self.episodes_reward = []
        #
        for ep in tqdm(list(range(self.num_episodes))):
```

```
            #
            state = self.get_state(self.env.reset())
            #
            done = False
            #
            truncated = False
            #
            tot_rew = 0

            #             Q-
            if self.eps > self.eps_threshold:
                self.eps -= self.eps_decay

            #
            while not (done or truncated):
                #
                #   SARSA
                action = self.make_action(state)

                #
                next_state, rew, done, truncated, _ = self.env.step(action)

                #             Q    SARSA (        )
                # self.Q[state][action] = self.Q[state][action] + self.lr * \
                #     (rew + self.gamma * self.Q[next_state][next_action] -␣
↪self.Q[state][action])

                #               Q-
                self.Q[state][action] = self.Q[state][action] + self.lr * \
                    (rew + self.gamma * np.max(self.Q[next_state]) - self.
↪Q[state][action])

                #
                state = next_state
                #
                tot_rew += rew
                if (done or truncated):
                    self.episodes_reward.append(tot_rew)
```

0.0.4 Q-

[ ]:
```
class DoubleQLearning_Agent(BasicAgent):
    '''
                Double Q-Learning
    '''
    #
    ALGO_NAME = '     Q-     '
```

```
def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
    #
    super().__init__(env, eps)
    #
    self.Q2 = np.zeros((self.nS, self.nA))
    # Learning rate
    self.lr=lr
    #
    self.gamma = gamma
    #
    self.num_episodes=num_episodes
    #               eps
    self.eps_decay=0.00005
    self.eps_threshold=0.01


def greedy(self, state):
    '''
    <<   >>
              ,                  Q-
          state
    '''
    temp_q = self.Q[state] + self.Q2[state]
    return np.argmax(temp_q)

def print_q(self):
    print(f"    Q-             {self.ALGO_NAME}")
    print('Q1')
    print(self.Q)
    print('Q2')
    print(self.Q2)

def learn(self):
    '''
                    Double Q-Learning
    '''
    self.episodes_reward = []
    #
    for ep in tqdm(list(range(self.num_episodes))):
        #
        state = self.get_state(self.env.reset())
        #
        done = False
        #
        truncated = False
        #
```

```
            tot_rew = 0

            #             Q-
            if self.eps > self.eps_threshold:
                self.eps -= self.eps_decay

            #
            while not (done or truncated):
                #
                #   SARSA
                action = self.make_action(state)

                #
                next_state, rew, done, truncated, _ = self.env.step(action)

                if np.random.rand() < 0.5:
                    #
                    self.Q[state][action] = self.Q[state][action] + self.lr * \
                        (rew + self.gamma * self.Q2[next_state][np.argmax(self.
↪Q[next_state])] - self.Q[state][action])
                else:
                    #
                    self.Q2[state][action] = self.Q2[state][action] + self.lr *␣
↪\
                        (rew + self.gamma * self.Q[next_state][np.argmax(self.
↪Q2[next_state])] - self.Q2[state][action])

                #
                state = next_state
                #
                tot_rew += rew
                if (done or truncated):
                    self.episodes_reward.append(tot_rew)
```

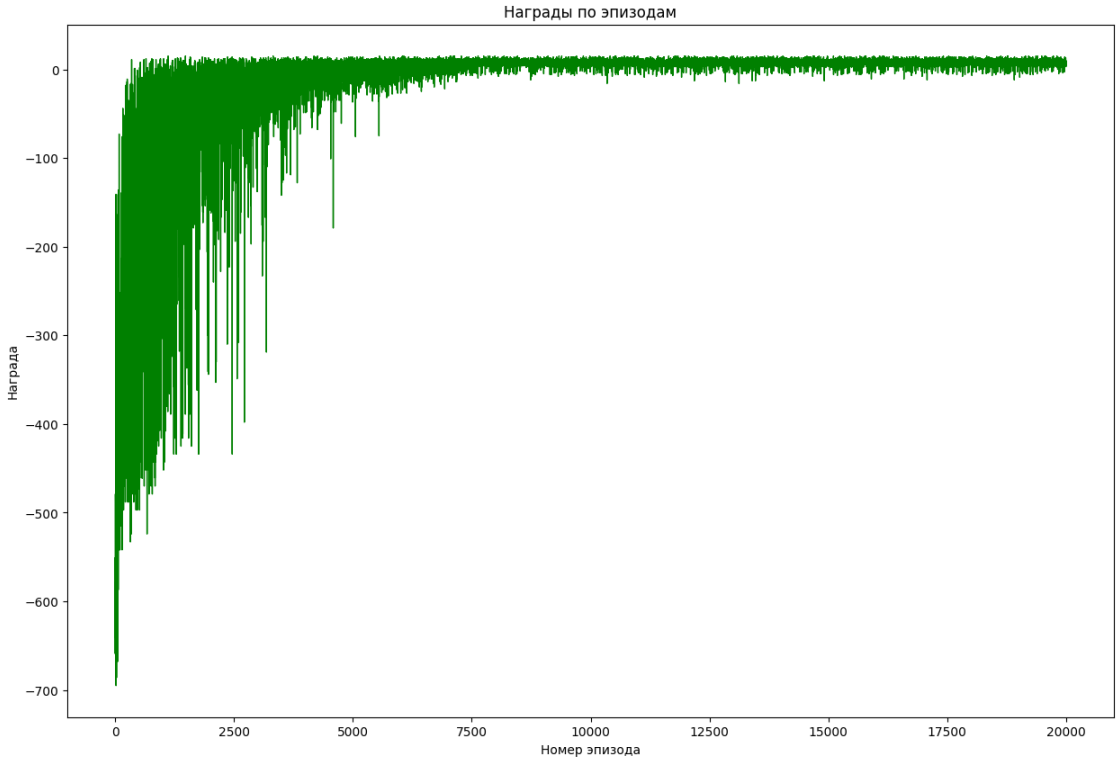
[ ]:
```
def play_agent(agent):
    '''

    '''
    env2 = gym.make('Taxi-v3', render_mode='human')
    state = env2.reset()[0]
    done = False
    while not done:
        action = agent.greedy(state)
        next_state, reward, terminated, truncated, _ = env2.step(action)
        env2.render()
        state = next_state
        if terminated or truncated:
```

```
            done = True

def run_sarsa():
    env = gym.make('Taxi-v3')
    agent = SARSA_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

def run_q_learning():
    env = gym.make('Taxi-v3')
    agent = QLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)

def run_double_q_learning():
    env = gym.make('Taxi-v3')
    agent = DoubleQLearning_Agent(env)
    agent.learn()
    agent.print_q()
    agent.draw_episodes_reward()
    play_agent(agent)
```

0.0.5 SARSA: eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000

[ ]:
```
run_sarsa()
```

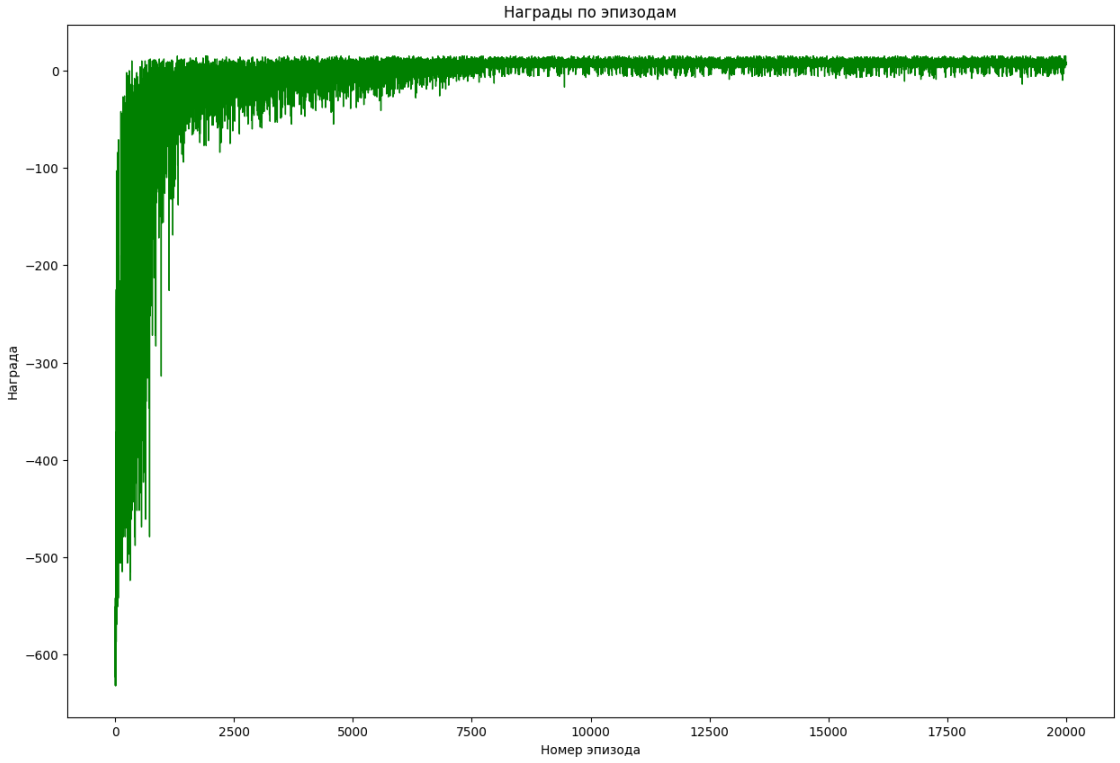
```
  0%|          | 0/20000 [00:00<?, ?it/s]100%|      | 20000/20000
[00:03<00:00, 6075.86it/s]

    Q-              SARSA
[[  0.           0.           0.           0.           0.
    0.        ]
 [ -9.39511871  -7.82633886  -4.61973996  -7.91608244   7.52831391
  -14.58270727]
 [  0.72064374   2.33945919   2.24226059   1.10286948  13.0813314
   -6.7719644 ]
 …
 [ -3.08864603   5.58824812  -3.13475835  -2.35462461  -7.02756406
   -9.25639657]
 [ -7.77032983  -8.05657849  -8.49347554  -1.95002266 -14.61611739
  -14.22935898]
 [  5.45984704   4.18158543  10.07790051  18.007744     1.41430917
    1.34707593]]
```



0.0.6 Q- : eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000

[ ]:
```
run_q_learning()
```

```
100%|      | 20000/20000 [00:03<00:00, 5488.88it/s]

    Q-              Q-
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 5.55943204  6.43481406  4.66190971  6.25449521  8.36234335 -2.09352707]
 [ 9.16004638 11.34118969  9.82937408 10.403992   13.27445578  2.29056275]
 …
 [ 1.48773717 13.44572688  1.69519677  0.142458   -3.59137074 -5.56482255]
 [-1.95348732  8.73563506 -2.38814501  0.59149548 -8.87902035 -7.38586058]
 [ 0.          6.46636365  9.54364847 18.59693097  0.87934239  3.81148721]]
```



0.0.7 Q- : eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000

[ ]:
```
run_double_q_learning()
```

```
100%|      | 20000/20000 [00:04<00:00, 4883.75it/s]

    Q-                  Q-
Q1
[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 1.16163046e+00  6.45016960e-01  1.00769948e+00  3.72291477e+00
   8.36234335e+00 -5.01468586e+00]
 [ 6.09505909e+00  8.36560187e+00 -4.07224966e-01  7.05822996e+00
   1.32744558e+01  6.45841012e-01]
 …
 [-7.99180545e-01  1.07550337e+01 -1.44867522e+00 -2.35590222e+00
  -5.86881524e+00 -5.62712420e+00]
 [-3.98247635e+00  4.37814489e+00 -3.92624883e+00 -2.60475579e+00
  -5.10770158e+00 -7.54219253e+00]
 [ 2.78683960e+00  2.85534245e+00  0.00000000e+00  1.83727537e+01
   4.18884557e-01 -7.50927475e-03]]
Q2
[[ 0.          0.          0.          0.          0.          0.        ]
 [-0.2934294   3.26994885 -1.42217566  5.04477693  8.36234335 -5.90593791]
```

```
[ 5.90112017  5.46169946  2.42918746  3.80673123 13.27445578 -0.5747265 ]
…
[-2.10538343 11.9391959  -1.57538495 -1.53177373 -3.18288299 -4.73475405]
[-3.74525344  3.01047271 -3.83516418 -4.04781581 -4.80672105 -5.4513151 ]
[ 3.14380145  1.05446787  1.51034447 18.44538414 -3.68630097  0.42193263]]
```