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

Лабораторная работа №5  
по дисциплине  
«Методы машинного обучения»  
на тему

**«**Обучение на основе временных различий**»**

Выполнил:  
студент группы ИУ5И-21М  
Фань Лицзе

Москва — 2024 г.

**1. Цель лабораторной работы**

ознакомление с базовыми методами обучения с подкреплением на основе временных различий.

**2. 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).

1. **Текст программы**

import numpy as np

import gym

from collections import defaultdict

import matplotlib.pyplot as plt

from tqdm import tqdm

class BasicAgent:

ALGO\_NAME = '---'

def \_\_init\_\_(self, env, eps=0.1):

self.env = env

self.nA = env.action\_space.n

# Используем defaultdict для удобства

self.Q = defaultdict(lambda: np.zeros(self.nA))

self.eps = eps

self.episodes\_reward = []

def get\_state(self, state):

# В CartPole состояния могут быть непрерывными, здесь мы их дискретизируем

# Например, округлим значения до определенной точности

return tuple([round(s, 2) for s in state])

def greedy(self, state):

state = self.get\_state(state)

return np.argmax(self.Q[state])

def make\_action(self, state):

state = self.get\_state(state)

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

return self.env.action\_space.sample()

else:

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('Награды по эпизодам')

plt.xlabel('Номер эпизода')

plt.ylabel('Награда')

plt.show()

def learn(self):

pass

  class SarsaAgent(BasicAgent):

ALGO\_NAME = 'SARSA'

def \_\_init\_\_(self, env, eps=0.1, lr=0.1, gamma=0.95, num\_episodes=2000):

super().\_\_init\_\_(env, eps)

self.lr = lr

self.gamma = gamma

self.num\_episodes = num\_episodes

def learn(self):

for ep in tqdm(range(self.num\_episodes)):

state = self.env.reset()

action = self.make\_action(state)

done = False

total\_reward = 0

while not done:

next\_state, reward, done, \_ = self.env.step(action)

next\_action = self.make\_action(next\_state)

self.Q[self.get\_state(state)][action] += self.lr \* (

reward + self.gamma \* self.Q[self.get\_state(next\_state)][next\_action] - self.Q[self.get\_state(state)][action]

)

state = next\_state

action = next\_action

total\_reward += reward

self.episodes\_reward.append(total\_reward)

     class QLearningAgent(BasicAgent):

ALGO\_NAME = 'Q-learning'

def \_\_init\_\_(self, env, eps=0.1, lr=0.1, gamma=0.95, num\_episodes=2000):

super().\_\_init\_\_(env, eps)

self.lr = lr

self.gamma = gamma

self.num\_episodes = num\_episodes

def learn(self):

for ep in tqdm(range(self.num\_episodes)):

state = self.env.reset()

done = False

total\_reward = 0

while not done:

action = self.make\_action(state)

next\_state, reward, done, \_ = self.env.step(action)

self.Q[self.get\_state(state)][action] += self.lr \* (

reward + self.gamma \* np.max(self.Q[self.get\_state(next\_state)]) - self.Q[self.get\_state(state)][action]

)

state = next\_state

total\_reward += reward

self.episodes\_reward.append(total\_reward)

   class DoubleQLearningAgent(BasicAgent):

ALGO\_NAME = 'Double Q-learning'

def \_\_init\_\_(self, env, eps=0.1, lr=0.1, gamma=0.95, num\_episodes=2000):

super().\_\_init\_\_(env, eps)

self.lr = lr

self.gamma = gamma

self.num\_episodes = num\_episodes

# 使用两个Q字典来实现双Q学习

self.Q1 = defaultdict(lambda: np.zeros(self.nA))

self.Q2 = defaultdict(lambda: np.zeros(self.nA))

def make\_action(self, state):

state = self.get\_state(state)

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

return self.env.action\_space.sample()

else:

# 使用Q1和Q2的和来选择行动

return np.argmax(self.Q1[state] + self.Q2[state])

def learn(self):

for ep in tqdm(range(self.num\_episodes)):

state = self.env.reset()

done = False

total\_reward = 0

while not done:

action = self.make\_action(state)

next\_state, reward, done, \_ = self.env.step(action)

if np.random.uniform(0, 1) < 0.5:

best\_next\_action = np.argmax(self.Q1[self.get\_state(next\_state)])

self.Q1[self.get\_state(state)][action] += self.lr \* (

reward + self.gamma \* self.Q2[self.get\_state(next\_state)][best\_next\_action] - self.Q1[self.get\_state(state)][action]

)

else:

best\_next\_action = np.argmax(self.Q2[self.get\_state(next\_state)])

self.Q2[self.get\_state(state)][action] += self.lr \* (

reward + self.gamma \* self.Q1[self.get\_state(next\_state)][best\_next\_action] - self.Q2[self.get\_state(state)][action]

)

state = next\_state

total\_reward += reward

self.episodes\_reward.append(total\_reward)

  def play\_agent(agent):

env2 = gym.make('CartPole-v1', render\_mode='human')

state = env2.reset()

done = False

while not done:

action = agent.greedy(state)

next\_state, reward, done, info = env2.step(action)

env2.render()

state = next\_state

if done:

break

def run\_sarsa():

env = gym.make('CartPole-v1')

agent = SarsaAgent(env)

agent.learn()

agent.draw\_episodes\_reward()

play\_agent(agent)

 def run\_q\_learning():

env = gym.make('CartPole-v1')

agent = QLearningAgent(env)

agent.learn()

agent.draw\_episodes\_reward()

play\_agent(agent)

def run\_double\_q\_learning():

env = gym.make('CartPole-v1')

agent = DoubleQLearningAgent(env)

agent.learn()

agent.draw\_episodes\_reward()

play\_agent(agent)

      def main():

# run\_sarsa()

# run\_q\_learning()

run\_double\_q\_learning()

if \_\_name\_\_ == '\_\_main\_\_':

main()

1. **Результат**

