

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
им. Н.Э. Баумана

Факультет «Информатика и системы управления»  
Кафедра «Систем обработки информации и управления»

**Рубежный контроль № 2**  
по дисциплине «Методы машинного обучения»

ИСПОЛНИТЕЛЬ:  
группа ИУ5-24М

Поташников М.Д.  
ФИО

Москва - 2023

---

## 1. Задание

Для одного из алгоритмов временных различий, реализованных Вами в соответствующей лабораторная работе:

- SARSA
- Q-обучение
- Двойное Q-обучение

осуществите подбор гиперпараметров. Критерием оптимизации должна являться суммарная награда.

## 2. Текст программы

```
1  #!/usr/bin/env python
2
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import gymnasium as gym
6  from tqdm import tqdm
7  import matplotlib
8  matplotlib.use('TkAgg')
9
10 # ***** БАЗОВЫЙ АГЕНТ *****
11
12 class BasicAgent:
13     '''
14     Базовый агент, от которого наследуются стратегии обучения
15     '''
16
17     # Наименование алгоритма
18     ALGO_NAME = '___'
19
20     def __init__(self, env, eps=0.1):
21         # Среда
22         self.env = env
23         # Размерности Q-матрицы
24         self.nA = env.action_space.n
25         self.nS = env.observation_space.n
26         #и сама матрица
27         self.Q = np.zeros((self.nS, self.nA))
28         # Значения коэффициентов
29         # Порог выбора случайного действия
30         self.eps=eps
31         # Награды по эпизодам
32         self.episodes_reward = []
33
34     def print_q(self):
35         print('Вывод Q-матрицы для алгоритма ', self.ALGO_NAME)
36         print(self.Q)
37
38     def get_state(self, state):
39         '''
40         Возвращает правильное начальное состояние
41         '''
42         if type(state) is tuple:
43             # Если состояние вернулось с виде кортежа, то вернуть только номер состояния
```

```

44         return state[0]
45     else:
46         return state
47
48 def greedy(self, state):
49     '''
50     <<Жадное>> текущее действие
51     Возвращает действие, соответствующее максимальному Q-значению
52     для состояния state
53     '''
54     return np.argmax(self.Q[state])
55
56 def make_action(self, state):
57     '''
58     Выбор действия агентом
59     '''
60     if np.random.uniform(0,1) < self.eps:
61
62         # Если вероятность меньше eps
63         # то выбирается случайное действие
64         return self.env.action_space.sample()
65     else:
66         # иначе действие, соответствующее максимальному Q-значению
67         return self.greedy(state)
68
69 def draw_episodes_reward(self):
70     # Построение графика наград по эпизодам
71     fig, ax = plt.subplots(figsize = (15,10))
72     y = self.episodes_reward
73     x = list(range(1, len(y)+1))
74     plt.plot(x, y, '-', linewidth=1, color='green')
75     plt.title('Награды по эпизодам')
76     plt.xlabel('Номер эпизода')
77     plt.ylabel('Награда')
78     plt.show()
79
80 def learn():
81     '''
82     Реализация алгоритма обучения
83     '''
84     pass
85
86 # ***** SARSA *****
87
88 class SARSA_Agent(BasicAgent):
89     '''
90     Реализация алгоритма SARSA
91     '''
92     # Наименование алгоритма
93     ALGO_NAME = 'SARSA'
94
95     def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
96         # Вызов конструктора верхнего уровня
97         super().__init__(env, eps)
98         # Learning rate
99         self.lr=lr
100        # Коэффициент дисконтирования
101        self.gamma = gamma
102        # Количество эпизодов
103        self.num_episodes=num_episodes
104        # Постепенное уменьшение eps

```

```

105     self.eps_decay=0.00005
106     self.eps_threshold=0.01
107
108     def learn(self):
109         '''
110         Обучение на основе алгоритма SARSA
111         '''
112         self.episodes_reward = []
113         # Цикл по эпизодам
114         for ep in tqdm(list(range(self.num_episodes))):
115             # Начальное состояние среды
116             state = self.get_state(self.env.reset())
117             # Флаг штатного завершения эпизода
118             done = False
119             # Флаг нештатного завершения эпизода
120             truncated = False
121             # Суммарная награда по эпизоду
122             tot_rew = 0
123
124             # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия
125             if self.eps > self.eps_threshold:
126                 self.eps -= self.eps_decay
127
128             # Выбор действия
129             action = self.make_action(state)
130
131             # Проигрывание одного эпизода до финального состояния
132             while not (done or truncated):
133                 # Выполняем шаг в среде
134                 next_state, rew, done, truncated, _ = self.env.step(action)
135
136                 # Выполняем следующее действие
137                 next_action = self.make_action(next_state)
138
139                 # Правило обновления Q для SARSA
140                 self.Q[state][action] = self.Q[state][action] + self.lr * \
141                     (rew + self.gamma * self.Q[next_state][next_action] -
142                     self.Q[state][action])
143
144                 # Следующее состояние считаем текущим
145                 state = next_state
146                 action = next_action
147                 # Суммарная награда за эпизод
148                 tot_rew += rew
149                 if (done or truncated):
150                     self.episodes_reward.append(tot_rew)
151 # ***** Q-обучение *****
152
153     class QLearning_Agent(BasicAgent):
154         '''
155         Реализация алгоритма Q-Learning
156         '''
157         # Наименование алгоритма
158         ALGO_NAME = 'Q-обучение'
159
160         def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
161             # Вызов конструктора верхнего уровня
162             super().__init__(env, eps)
163             # Learning rate
164             self.lr=lr

```

```

165     # Коэффициент дисконтирования
166     self.gamma = gamma
167     # Количество эпизодов
168     self.num_episodes=num_episodes
169     # Постепенное уменьшение eps
170     self.eps_decay=0.00005
171     self.eps_threshold=0.01
172
173     def learn(self):
174         '''
175         Обучение на основе алгоритма Q-Learning
176         '''
177         self.episodes_reward = []
178         # Цикл по эпизодам
179         for ep in tqdm(list(range(self.num_episodes))):
180             # Начальное состояние среды
181             state = self.get_state(self.env.reset())
182             # Флаг штатного завершения эпизода
183             done = False
184             # Флаг нештатного завершения эпизода
185             truncated = False
186             # Суммарная награда по эпизоду
187             tot_rew = 0
188
189             # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия
190             if self.eps > self.eps_threshold:
191                 self.eps -= self.eps_decay
192
193             # Проигрывание одного эпизода до финального состояния
194             while not (done or truncated):
195                 # Выбор действия
196                 # В SARSA следующее действие выбиралось после шага в среде
197                 action = self.make_action(state)
198
199                 # Выполняем шаг в среде
200                 next_state, rew, done, truncated, _ = self.env.step(action)
201
202                 # Правило обновления Q для SARSA (для сравнения)
203                 # self.Q[state][action] = self.Q[state][action] + self.lr * \
204                 #     (rew + self.gamma * self.Q[next_state][next_action] -
205                 #     self.Q[state][action])
206
207                 # Правило обновления для Q-обучения
208                 self.Q[state][action] = self.Q[state][action] + self.lr * \
209                     (rew + self.gamma * np.max(self.Q[next_state]) - self.Q[state][action])
210
211                 # Следующее состояние считаем текущим
212                 state = next_state
213                 # Суммарная награда за эпизод
214                 tot_rew += rew
215                 if (done or truncated):
216                     self.episodes_reward.append(tot_rew)
217
218     # ***** Двойное Q-обучение *****
219     class DoubleQLearning_Agent(BasicAgent):
220         '''
221         Реализация алгоритма Double Q-Learning
222         '''
223         # Наименование алгоритма

```

```

224 ALGO_NAME = 'Двойное Q-обучение'
225
226 def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
227     # Вызов конструктора верхнего уровня
228     super().__init__(env, eps)
229     # Вторая матрица
230     self.Q2 = np.zeros((self.nS, self.nA))
231     # Learning rate
232     self.lr=lr
233     # Коэффициент дисконтирования
234     self.gamma = gamma
235     # Количество эпизодов
236     self.num_episodes=num_episodes
237     # Постепенное уменьшение eps
238     self.eps_decay=0.00005
239     self.eps_threshold=0.01
240
241
242 def greedy(self, state):
243     '''
244     <<Жадное>> текущее действие
245     Возвращает действие, соответствующее максимальному Q-значению
246     для состояния state
247     '''
248     temp_q = self.Q[state] + self.Q2[state]
249     return np.argmax(temp_q)
250
251 def print_q(self):
252     print(f"Вывод Q-матриц для алгоритма {self.ALGO_NAME}")
253     print('Q1')
254     print(self.Q)
255     print('Q2')
256     print(self.Q2)
257
258 def learn(self):
259     '''
260     Обучение на основе алгоритма Double Q-Learning
261     '''
262     self.episodes_reward = []
263     # Цикл по эпизодам
264     for ep in tqdm(list(range(self.num_episodes))):
265         # Начальное состояние среды
266         state = self.get_state(self.env.reset())
267         # Флаг штатного завершения эпизода
268         done = False
269         # Флаг нештатного завершения эпизода
270         truncated = False
271         # Суммарная награда по эпизоду
272         tot_rew = 0
273
274         # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия
275         if self.eps > self.eps_threshold:
276             self.eps -= self.eps_decay
277
278         # Проигрывание одного эпизода до финального состояния
279         while not (done or truncated):
280             # Выбор действия
281             # В SARSA следующее действие выбиралось после шага в среде
282             action = self.make_action(state)
283
284             # Выполняем шаг в среде

```

```

285         next_state, rew, done, truncated, _ = self.env.step(action)
286
287         if np.random.rand() < 0.5:
288             # Обновление первой таблицы
289             self.Q[state][action] = self.Q[state][action] + self.lr * \
290                 (rew + self.gamma * self.Q2[next_state][np.argmax(self.Q[next_state])]
291                  - self.Q[state][action])
292         else:
293             # Обновление второй таблицы
294             self.Q2[state][action] = self.Q2[state][action] + self.lr * \
295                 (rew + self.gamma * self.Q[next_state][np.argmax(self.Q2[next_state])]
296                  - self.Q2[state][action])
297
298             # Следующее состояние считаем текущим
299             state = next_state
300             # Суммарная награда за эпизод
301             tot_rew += rew
302             if (done or truncated):
303                 self.episodes_reward.append(tot_rew)
304
305 def play_agent(agent):
306     '''
307     Проигрывание сессии для обученного агента
308     '''
309     env2 = gym.make('Taxi-v3', render_mode='human')
310     state = env2.reset()[0]
311     done = False
312     while not done:
313         action = agent.greedy(state)
314         next_state, reward, terminated, truncated, _ = env2.step(action)
315         env2.render()
316         state = next_state
317         if terminated or truncated:
318             done = True
319
320 def plot_rewards(x, y):
321     # Построение графика наград по эпизодам
322     fig, ax = plt.subplots(figsize = (15,10))
323     plt.plot(x, y, '-', linewidth=1, color='green')
324     plt.title('Награды')
325     plt.xlabel('Параметр')
326     plt.ylabel('Награда')
327     plt.show()
328
329 def bruteforce_sarsa():
330     env = gym.make('Taxi-v3')
331     rewards_eps = []
332     rewards_lr = []
333     rewards_gamma = []
334     x = np.arange(0.1, 1, 0.1)
335     for i in x:
336         agent = SARSA_Agent(env,eps=i)
337         agent.learn()
338         agent.print_q()
339         rewards_eps.append(np.asarray(agent.episodes_reward).sum())
340     plot_rewards(x, rewards_eps)
341     best_eps = x[rewards_eps.index(max(rewards_eps))]
342     print(f"Best eps: {best_eps}")
343     x = np.arange(0, 1, 0.03)
344     for i in x:
345         agent = SARSA_Agent(env, eps = best_eps, lr = i)

```

```

344     agent.learn()
345     agent.print_q()
346     rewards_lr.append(np.asarray(agent.episodes_reward).sum())
347     best_lr = x[rewards_lr.index(max(rewards_lr))]
348     print(f"Best lr: {best_lr}")
349     plot_rewards(x, rewards_lr)
350     x = np.arange(0, 1, 0.03)
351     for i in x:
352         agent = SARSA_Agent(env, eps = best_eps, lr = best_lr, gamma = i)
353         agent.learn()
354         agent.print_q()
355         rewards_gamma.append(np.asarray(agent.episodes_reward).sum())
356         best_gamma = x[rewards_gamma.index(max(rewards_gamma))]
357         print(f"Best gamma: {best_gamma}")
358         plot_rewards(x, rewards_gamma)
359     print(rewards_eps)
360     print(rewards_lr)
361     print(rewards_gamma)
362     print(f"Best params: eps={best_eps}, lr={best_lr}, gamma={best_gamma}")
363
364 def run_sarsa():
365     env = gym.make('Taxi-v3')
366     agent = SARSA_Agent(env, eps=0.1, lr=0.39, gamma=0.93)
367     agent.learn()
368     agent.print_q()
369     agent.draw_episodes_reward()
370     play_agent(agent)
371
372 def run_q_learning():
373     env = gym.make('Taxi-v3')
374     agent = QLearning_Agent(env)
375     agent.learn()
376     agent.print_q()
377     agent.draw_episodes_reward()
378     play_agent(agent)
379
380 def run_double_q_learning():
381     env = gym.make('Taxi-v3')
382     agent = DoubleQLearning_Agent(env)
383     agent.learn()
384     agent.print_q()
385     agent.draw_episodes_reward()
386     play_agent(agent)
387
388 def main():
389     brute_force_sarsa()
390     # run_sarsa()
391     # run_q_learning()
392     # run_double_q_learning()
393
394 if __name__ == '__main__':
395     main()

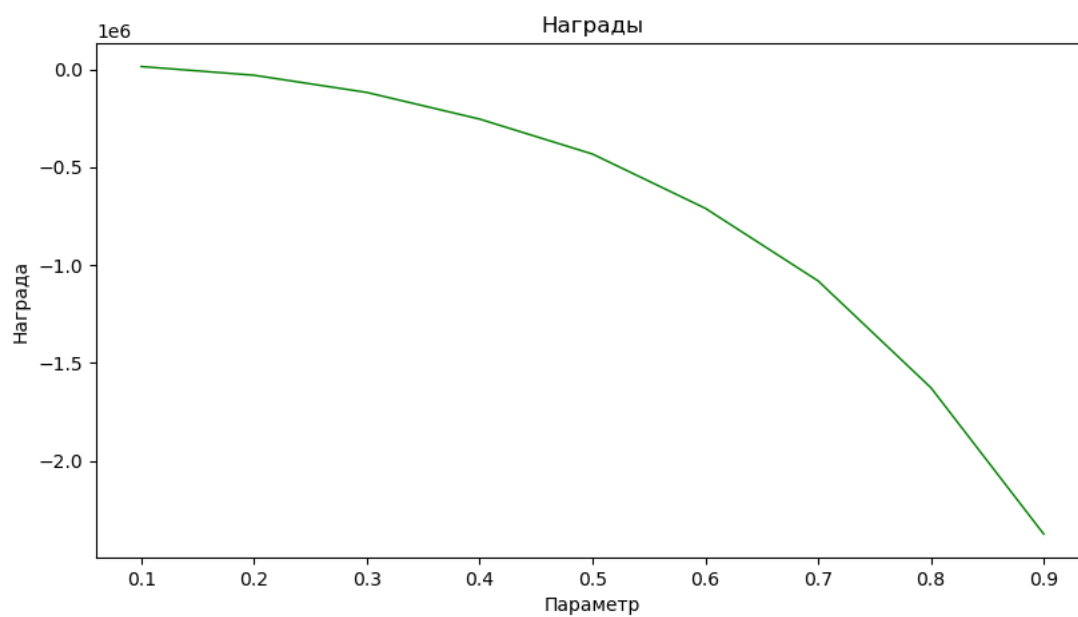
```

Программа производит автоматический перебор параметров и находит оптимальные.

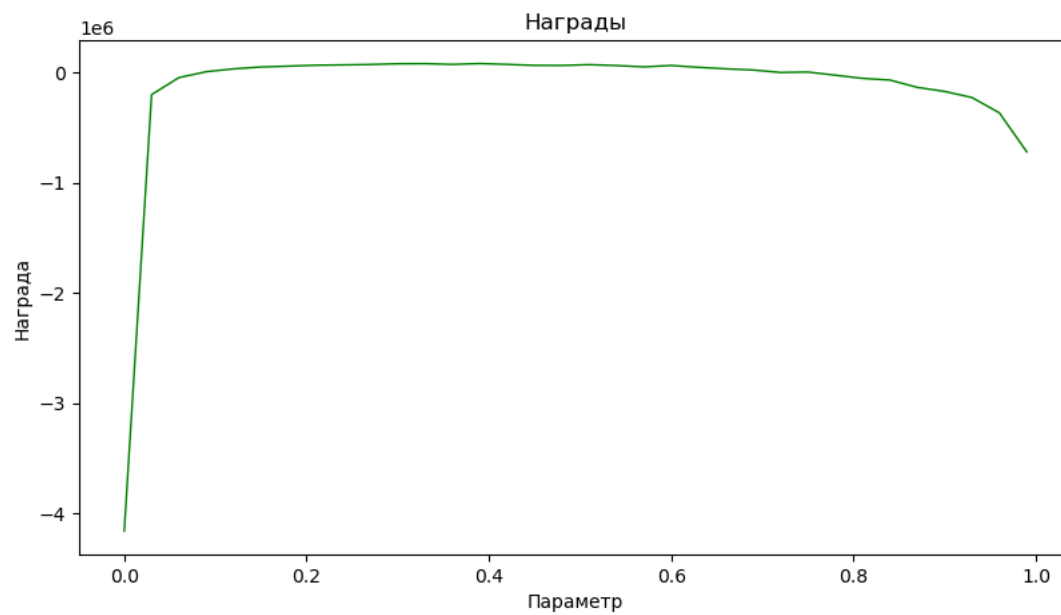


### 3. Графики

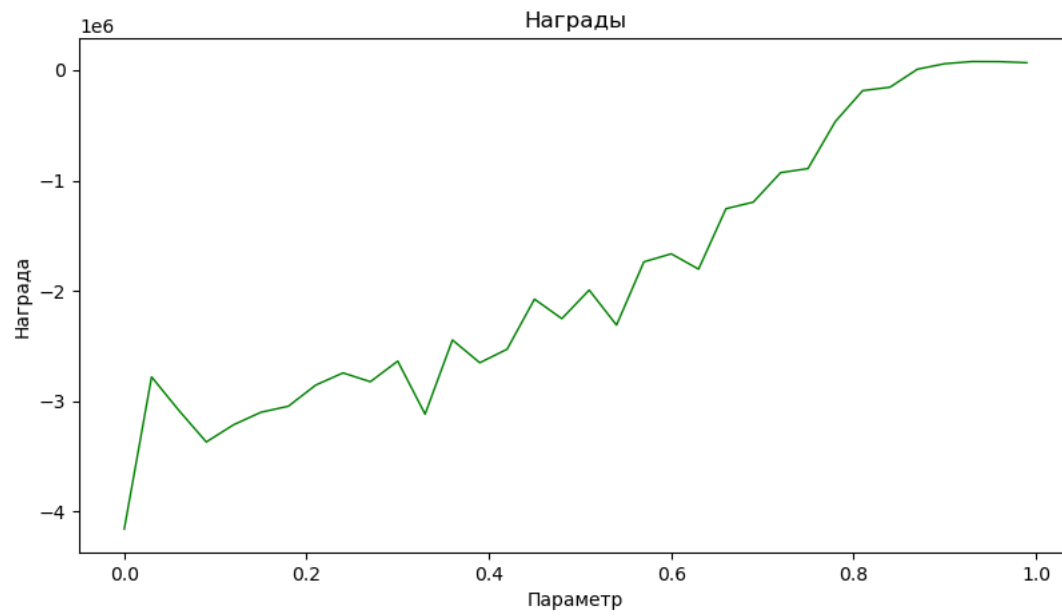
Параметр  $\epsilon$ :



Параметр  $lr$ :



Параметр gamma:



Лучшие параметры:  $\text{eps}=0.1$ ,  $\text{lr}=0.39$ ,  $\text{gamma}=0.93$

