



Министерство науки и высшего образования Российской
Федерации
Федеральное государственное бюджетное образовательное
учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления

Отчет по лабораторной работе №5
«Обучение на основе временных различий»

Студент группы ИУ5-25М

Зозуля О.А.

Москва, 2023 г.

Цель работы

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

Задание

На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

- SARSA
- Q-обучение
- Двойное Q-обучение

для любой среды обучения с подкреплением из библиотеки Gym.

Описание выполнения

```
1 import gym
2 from pprint import pprint
3
4 def main():
5     state, action = 0, 0
6     env = gym.make("CliffWalking-v0")
7     print('Пространство состояний:')
8     pprint(env.observation_space)
9     print()
10    print('Пространство действий:')
11    pprint(env.action_space)
12    print()
13    print('Диапазон наград:')
14    pprint(env.reward_range)
15    print()
16    print('Вероятности для 0 состояния и 0 действия:')
17    pprint(env.P[state][action])
18    print()
19    print('Вероятности для 0 состояния:')
20    pprint(env.P[state])
21
22
23 if __name__ == '__main__':
24     main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Пространство действий:
Discrete(4)

Диапазон наград:
(-inf, inf)

Вероятности для 0 состояния и 0 действия:
[(1.0, 0, -1, False)]

Вероятности для 0 состояния:
{0: [(1.0, 0, -1, False)],
1: [(1.0, 1, -1, False)],
2: [(1.0, 12, -1, False)],
3: [(1.0, 0, -1, False)]}

PS C:\Users\Alexey>

Рисунок 1 – Информация о среде

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import gym
4 from tqdm import tqdm
5
6 # ***** БАЗОВЫЙ АГЕНТ *****
7
8 class BasicAgent:
9     """
10     Базовый агент, от которого наследуются стратегии обучения
11     """
12
13     # Наименование алгоритма
14     ALGO_NAME = '----'
15
16     def __init__(self, env, eps=0.1):
17         # Среда
18         self.env = env
19         # Размеры Q-матрицы
20         self.nA = env.action_space.n
21         self.nS = env.observation_space.n
22         # И сама матрица
23         self.Q = np.zeros((self.nS, self.nA))
24         # Значения коэффициентов
25         # Порог выбора случайного действия
26         self.eps = eps
27         # Награды по эпизодам
28         self.episodes_reward = []
29
30     def print_q(self):
31         print('Вывод Q-матрицы для алгоритма ', self.ALGO_NAME)
32         print(self.Q)
33
34     def get_state(self, state):
35         """
36         Возвращает правильное начальное состояние
37         """
38         if type(state) is tuple:
39             # Если состояние вернулось в виде кортежа, то вернуть только номер состояния
40             return state[0]
41         else:
42             return state
43
44     def greedy(self, state):
45         """
46         <<Жадное>> текущее действие
47         Возвращает действие, соответствующее максимальному Q-значению
48         для состояния state
49         """
50         return np.argmax(self.Q[state])
51
52     def make_action(self, state):
53         """
54         Выбор действия агентом
55         """
56         if np.random.uniform(0,1) < self.eps:
57             # Если вероятность меньше eps
58             # то выбирается случайное действие
59             return self.env.action_space.sample()
60         else:
61             # иначе действие, соответствующее максимальному Q-значению
62             return self.greedy(state)
63
64     def draw_episodes_reward(self):
65         # Построение графика наград по эпизодам
66         fig, ax = plt.subplots(figsize = (15,10))
67         y = self.episodes_reward
68         x = list(range(1, len(y)+1))
69         plt.plot(x, y, '-', linewidth=1, color='green')
70         plt.title('Награды по эпизодам')
71         plt.xlabel('Номер эпизода')
72         plt.ylabel('Награда')
73         plt.show()
74
75     def learn():
76         """
77         Реализация алгоритма обучения
78         """
79         pass

```

Рисунок 2-3 – Установка базовых параметров

```

90 class SARSA_Agent(BasicAgent):
91     """
92     Реализация алгоритма SARSA
93     """
94     # Наименование алгоритма
95     ALGO_NAME = 'SARSA'
96
97
98     def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
99         # Вызов конструктора верхнего уровня
100         super().__init__(env, eps)
101         # Learning rate
102         self.lr=lr
103         # Коэффициент дисконтирования
104         self.gamma = gamma
105         # Количество эпизодов
106         self.num_episodes=num_episodes
107         # Постепенное уменьшение eps
108         self.eps_decay=0.00005
109         self.eps_threshold=0.01
110
111
112     def learn(self):
113         """
114         Обучение на основе алгоритма SARSA
115         """
116         self.episodes_reward = []
117         # Цикл по эпизодам
118         for ep in tqdm(list(range(self.num_episodes))):
119             # Начальное состояние среды
120             state = self.get_state(self.env.reset())
121             # Флаг штатного завершения эпизода
122             done = False
123             # Флаг нештатного завершения эпизода
124             truncated = False
125             # Суммарная награда по эпизоду
126             tot_rew = 0
127
128             # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия
129             if self.eps > self.eps_threshold:
130                 self.eps -= self.eps_decay
131
132             # Выбор действия
133             action = self.make_action(state)
134
135
136             # Прогривание одного эпизода до финального состояния
137             while not (done or truncated):
138                 # Выполняем шаг в среде
139                 next_state, rew, done, truncated, _ = self.env.step(action)
140
141                 # Выполняем следующее действие
142                 next_action = self.make_action(next_state)
143
144                 # Правило обновления Q для SARSA
145                 self.Q[state][action] = self.Q[state][action] + self.lr * \
146                     (rew + self.gamma * self.Q[next_state][next_action] - self.Q[state][action])
147
148                 # Следующее состояние считаем текущим
149                 state = next_state
150                 action = next_action
151             # Суммарная награда за эпизод
152             tot_rew += rew
153             if (done or truncated):
154                 self.episodes_reward.append(tot_rew)

```

Рисунок 3-4 – Реализация алгоритма SARSA

```

158 class QLearning_Agent(BasicAgent):
159     """
160     Реализация алгоритма Q-Learning
161     """
162     # Наименование алгоритма
163     ALGO_NAME = 'Q-обучение'
164
165
166     def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
167         # Вызов конструктора верхнего уровня
168         super().__init__(env, eps)
169         # Learning rate
170         self.lr=lr
171         # Коэффициент дисконтирования
172         self.gamma = gamma
173         # Количество эпизодов
174         self.num_episodes=num_episodes
175         # Постепенное уменьшение eps
176         self.eps_decay=0.00005
177         self.eps_threshold=0.01
178
179
180     def learn(self):
181         """
182         Обучение на основе алгоритма Q-Learning
183         """
184         self.episodes_reward = []
185         # Цикл по эпизодам
186         for ep in tqdm(list(range(self.num_episodes))):
187             # Начальное состояние среды
188             state = self.get_state(self.env.reset())
189             # Флаг штатного завершения эпизода
190             done = False
191             # Флаг нештатного завершения эпизода
192             truncated = False
193             # Суммарная награда по эпизоду
194             tot_rew = 0
195
196             # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия
197             if self.eps > self.eps_threshold:
198                 self.eps -= self.eps_decay
199

```

```

200 # Програмирование одного эпизода до финального состояния
201 while not (done or truncated):
202
203     # Выбор действия
204     # В SARSA следующее действие выбиралось после шага в среде
205     action = self.make_action(state)
206
207     # Выполняем шаг в среде
208     next_state, rew, done, truncated, _ = self.env.step(action)
209
210     # Правило обновления Q для SARSA (для сравнения)
211     # self.Q[state][action] = self.Q[state][action] + self.lr * \
212     #     (rew + self.gamma * self.Q[next_state][next_action] - self.Q[state][action])
213
214     # Правило обновления для Q-обучения
215     self.Q[state][action] = self.Q[state][action] + self.lr * \
216     #     (rew + self.gamma * np.max(self.Q[next_state]) - self.Q[state][action])
217
218     # Следующее состояние считаем текущим
219     state = next_state
220     # Суммарная награда за эпизод
221     tot_rew += rew
222     if (done or truncated):
223         self.episodes_reward.append(tot_rew)

```

Рисунок 5-6 – Реализация Q-обучения

```

227 class DoubleQLearningAgent(BasicAgent):
228     """
229     Реализация алгоритма Double Q-Learning
230     """
231     # Наименование алгоритма
232     ALGO_NAME = 'Двойное Q-обучение'
233
234
235     def __init__(self, env, eps=0.4, lr=0.1, gamma=0.98, num_episodes=20000):
236         # Вызов конструктора верхнего уровня
237         super().__init__(env, eps)
238         # Вторая матрица
239         self.Q2 = np.zeros((self.nS, self.nA))
240         # Learning rate
241         self.lr=lr
242         # Коэффициент дисконтирования
243         self.gamma = gamma
244         # Количество эпизодов
245         self.num_episodes=num_episodes
246         # Постепенное уменьшение eps
247         self.eps_decay=0.00005
248         self.eps_threshold=0.01
249
250
251     def greedy(self, state):
252         """
253         <<Жадное>> текущее действие
254         Возвращает действие, соответствующее максимальному Q-значению
255         для состояния state
256         """
257         temp_q = self.Q[state] + self.Q2[state]
258         return np.argmax(temp_q)
259
260
261     def print_q(self):
262         print('Выход Q-матрицы для алгоритма ', self.ALGO_NAME)
263         print('Q1')
264         print(self.Q)
265         print('Q2')
266         print(self.Q2)

```

```

269     def learn(self):
270         """
271         Обучение на основе алгоритма Double Q-Learning
272         """
273         self.episodes_reward = []
274         # цикл по эпизодам
275         for ep in tqdm(list(range(self.num_episodes))):
276             # Начальное состояние среды
277             state = self.get_state(self.env.reset())
278             # флаг статического завершения эпизода
279             done = False
280             # флаг нештатного завершения эпизода
281             truncated = False
282             # Суммарная награда по эпизоду
283             tot_rew = 0
284
285             # По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия
286             if self.eps > self.eps_threshold:
287                 self.eps -= self.eps_decay
288
289             # Програмирование одного эпизода до финального состояния
290             while not (done or truncated):
291
292                 # Выбор действия
293                 # В SARSA следующее действие выбиралось после шага в среде
294                 action = self.make_action(state)
295
296                 # Выполняем шаг в среде
297                 next_state, rew, done, truncated, _ = self.env.step(action)
298
299                 if np.random.rand() < 0.5:
300                     # Обновление первой таблицы
301                     self.Q[state][action] = self.Q[state][action] + self.lr * \
302                     (rew + self.gamma * self.Q[next_state][np.argmax(self.Q[next_state])] - self.Q[state][action])
303                 else:
304                     # Обновление второй таблицы
305                     self.Q2[state][action] = self.Q2[state][action] + self.lr * \
306                     (rew + self.gamma * self.Q[next_state][np.argmax(self.Q2[next_state])] - self.Q2[state][action])
307
308             # Следующее состояние считаем текущим
309             state = next_state
310             # Суммарная награда за эпизод
311             tot_rew += rew
312             if (done or truncated):
313                 self.episodes_reward.append(tot_rew)

```

Рисунок 7-8 – Реализация двойного Q-обучения

```

315 def play_agent(agent):
316     ...
317     ...
318     Прогрессивная сессия для обучающего агента
319     ...
320     env2 = gym.make('CliffWalking-v0', render_mode='human')
321     state = env2.reset()[0]
322     done = False
323     while not done:
324         action = agent.greedy(state)
325         next_state, reward, terminated, truncated, _ = env2.step(action)
326         env2.render()
327         state = next_state
328         if terminated or truncated:
329             done = True
330     ...
331
332 def run_sarsa():
333     env = gym.make('CliffWalking-v0')
334     agent = SARSA_Agent(env)
335     agent.learn()
336     agent.print_q()
337     agent.draw_episodes_reward()
338     play_agent(agent)
339
340
341 def run_q_learning():
342     env = gym.make('CliffWalking-v0')
343     agent = Q_Learning_Agent(env)
344     agent.learn()
345     agent.print_q()
346     agent.draw_episodes_reward()
347     play_agent(agent)
348
349
350 def run_double_q_learning():
351     env = gym.make('CliffWalking-v0')
352     agent = DoubleQLearning_Agent(env)
353     agent.learn()
354     agent.print_q()
355     agent.draw_episodes_reward()
356     play_agent(agent)
357
358

```

Рисунок 9 – Инициализация алгоритмов

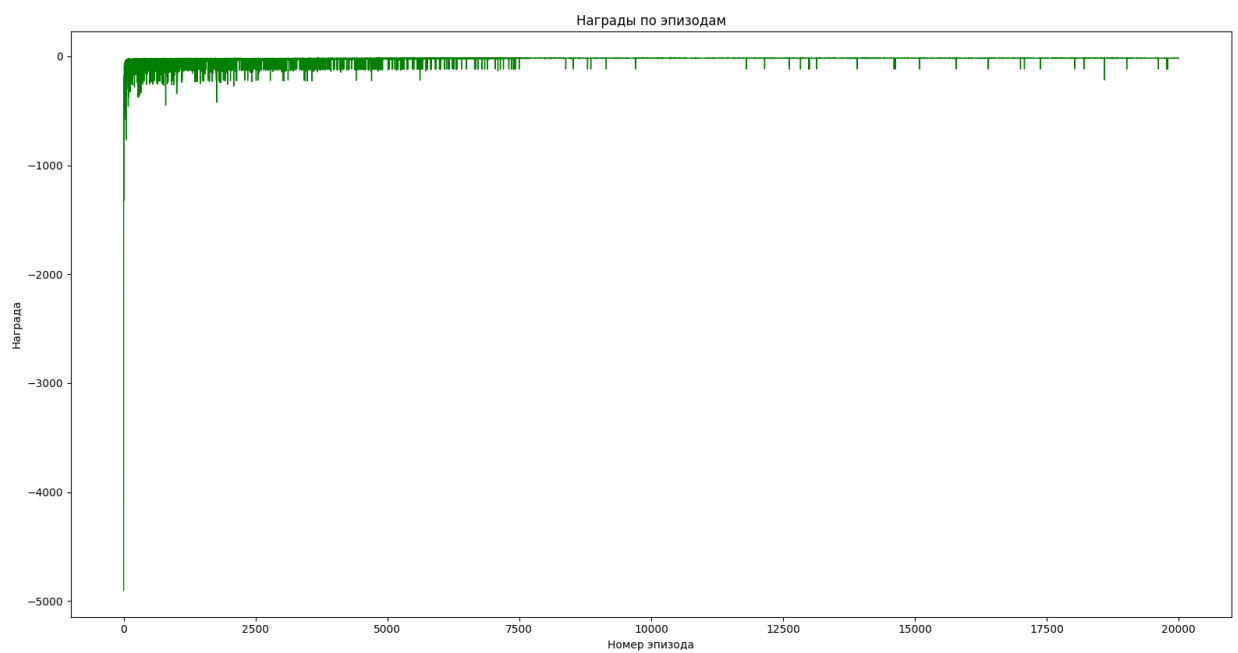


Рисунок 10 – Награды по эпизодам (SARSA)

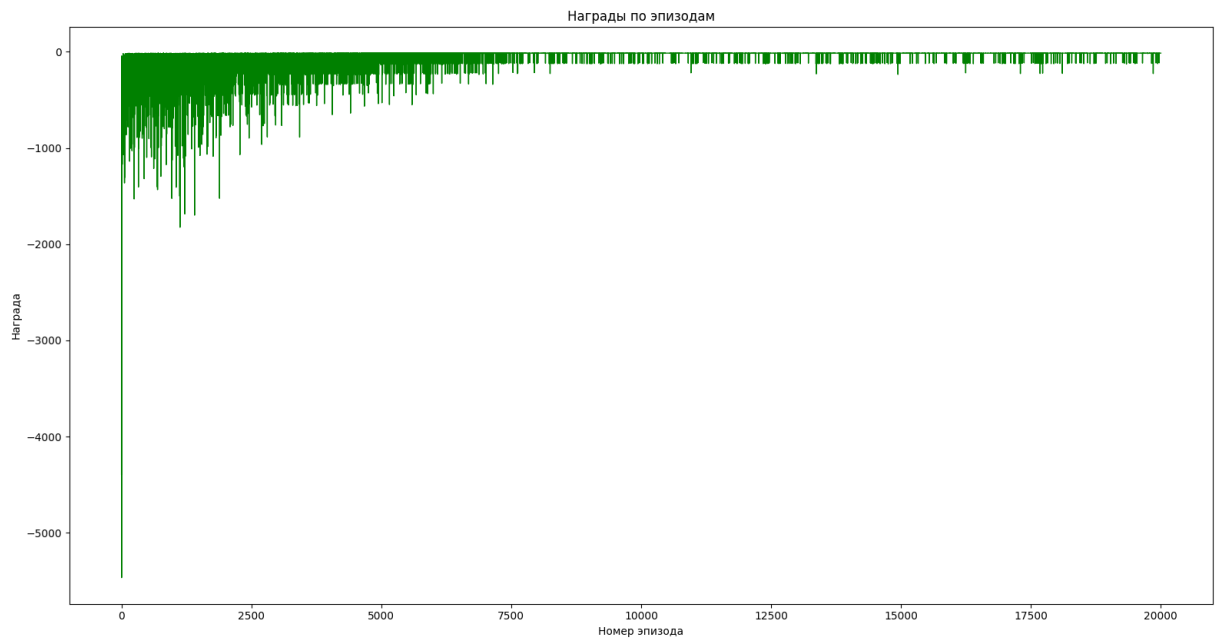


Рисунок 11 – Награды по эпизодам (Q-обучение)

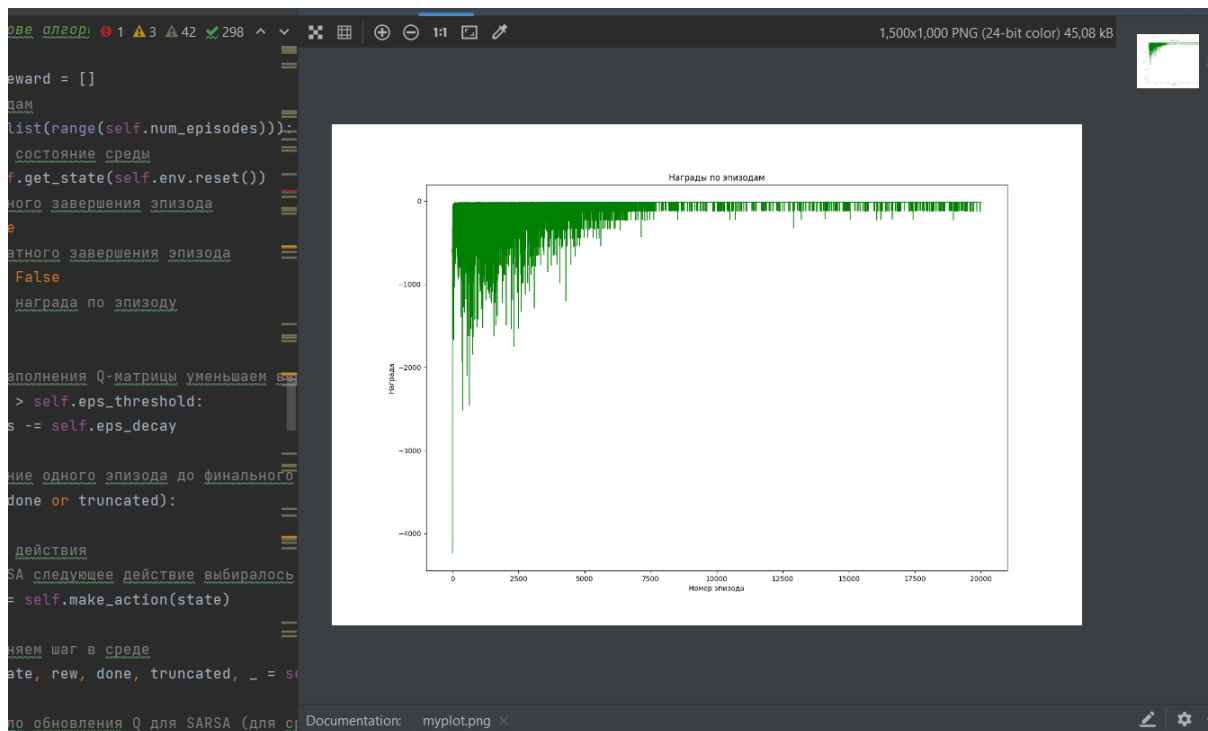


Рисунок 12 – Награды по эпизодам (Двойное Q-обучение)

Все алгоритмы смогли справиться с задачей нахождения маршрута в игре Cliff Walking, SARSA при этом нашла менее оптимальный путь, в то время как Q-алгоритмы нашли кратчайший.

Вывод

Таким образом, удалось реализовать алгоритмы SARSA, Q-обучения и двойного Q-обучения для среды обучения с подкреплением, таким образом ознакомившись с базовыми методами обучения с подкреплением на основе временных различий.