****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 временны’х различий»

Выполнил:  
студент группы ИУ5-23М  
Чжэн Сяохуэй

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](https://www.gymlibrary.dev/) (или аналогичной библиотеки).

**3. текст программы**

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** gym

**from** tqdm **import** tqdm

*# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* БАЗОВЫЙ АГЕНТ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

**class** BasicAgent:

'''

Базовый агент, от которого наследуются стратегии обучения

'''

*# Наименование алгоритма*

ALGO\_NAME **=** '---'

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

*# Среда*

self**.**env **=** env

*# Размерности Q-матрицы*

self**.**nA **=** env**.**action\_space**.**n

self**.**nS **=** env**.**observation\_space**.**n

*#и сама матрица*

self**.**Q **=** np**.**zeros((self**.**nS, self**.**nA))

*# Значения коэффициентов*

*# Порог выбора случайного действия*

self**.**eps**=**eps

*# Награды по эпизодам*

self**.**episodes\_reward **=** []

**def** print\_q(self):

print('Вывод Q-матрицы для алгоритма ', self**.**ALGO\_NAME)

print(self**.**Q)

**def** get\_state(self, state):

'''

Возвращает правильное начальное состояние

'''

**if** type(state) **is** tuple:

*# Если состояние вернулось с виде кортежа, то вернуть только номер состояния*

**return** state[0]

**else**:

**return** state

**def** greedy(self, state):

'''

<<Жадное>> текущее действие

Возвращает действие, соответствующее максимальному Q-значению

для состояния state

'''

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

**def** make\_action(self, state):

'''

Выбор действия агентом

'''

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

*# Если вероятность меньше eps*

*# то выбирается случайное действие*

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

**else**:

*# иначе действие, соответствующее максимальному Q-значению*

**return** self**.**greedy(state)

**def** draw\_episodes\_reward(self):

*# Построение графика наград по эпизодам*

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():

'''

Реализация алгоритма обучения

'''

**pass**

*# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SARSA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

**class** SARSA\_Agent(BasicAgent):

'''

Реализация алгоритма SARSA

'''

*# Наименование алгоритма*

ALGO\_NAME **=** 'SARSA'

**def** \_\_init\_\_(self, env, eps**=**0.4, lr**=**0.1, gamma**=**0.98, num\_episodes**=**20000):

*# Вызов конструктора верхнего уровня*

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

*# Learning rate*

self**.**lr**=**lr

*# Коэффициент дисконтирования*

self**.**gamma **=** gamma

*# Количество эпизодов*

self**.**num\_episodes**=**num\_episodes

*# Постепенное уменьшение eps*

self**.**eps\_decay**=**0.00005

self**.**eps\_threshold**=**0.01

**def** learn(self):

'''

Обучение на основе алгоритма SARSA

'''

self**.**episodes\_reward **=** []

*# Цикл по эпизодам*

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

*# Начальное состояние среды*

state **=** self**.**get\_state(self**.**env**.**reset())

*# Флаг штатного завершения эпизода*

done **=** **False**

*# Флаг нештатного завершения эпизода*

truncated **=** **False**

*# Суммарная награда по эпизоду*

tot\_rew **=** 0

*# По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия*

**if** self**.**eps **>** self**.**eps\_threshold:

self**.**eps **-=** self**.**eps\_decay

*# Выбор действия*

action **=** self**.**make\_action(state)

*# Проигрывание одного эпизода до финального состояния*

**while** **not** (done **or** truncated):

*# Выполняем шаг в среде*

next\_state, rew, done, truncated, \_ **=** self**.**env**.**step(action)

*# Выполняем следующее действие*

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

*# Правило обновления Q для SARSA*

self**.**Q[state][action] **=** self**.**Q[state][action] **+** self**.**lr **\*** \

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

*# Следующее состояние считаем текущим*

state **=** next\_state

action **=** next\_action

*# Суммарная награда за эпизод*

tot\_rew **+=** rew

**if** (done **or** truncated):

self**.**episodes\_reward**.**append(tot\_rew)

*# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Q-обучение \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

**class** QLearning\_Agent(BasicAgent):

'''

Реализация алгоритма Q-Learning

'''

*# Наименование алгоритма*

ALGO\_NAME **=** 'Q-обучение'

**def** \_\_init\_\_(self, env, eps**=**0.4, lr**=**0.1, gamma**=**0.98, num\_episodes**=**20000):

*# Вызов конструктора верхнего уровня*

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

*# Learning rate*

self**.**lr**=**lr

*# Коэффициент дисконтирования*

self**.**gamma **=** gamma

*# Количество эпизодов*

self**.**num\_episodes**=**num\_episodes

*# Постепенное уменьшение eps*

self**.**eps\_decay**=**0.00005

self**.**eps\_threshold**=**0.01

**def** learn(self):

'''

Обучение на основе алгоритма Q-Learning

'''

self**.**episodes\_reward **=** []

*# Цикл по эпизодам*

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

*# Начальное состояние среды*

state **=** self**.**get\_state(self**.**env**.**reset())

*# Флаг штатного завершения эпизода*

done **=** **False**

*# Флаг нештатного завершения эпизода*

truncated **=** **False**

*# Суммарная награда по эпизоду*

tot\_rew **=** 0

*# По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия*

**if** self**.**eps **>** self**.**eps\_threshold:

self**.**eps **-=** self**.**eps\_decay

*# Проигрывание одного эпизода до финального состояния*

**while** **not** (done **or** truncated):

*# Выбор действия*

*# В SARSA следующее действие выбиралось после шага в среде*

action **=** self**.**make\_action(state)

*# Выполняем шаг в среде*

next\_state, rew, done, truncated, \_ **=** self**.**env**.**step(action)

*# Правило обновления Q для SARSA (для сравнения)*

*# self.Q[state][action] = self.Q[state][action] + self.lr \* \*

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

*# Правило обновления для Q-обучения*

self**.**Q[state][action] **=** self**.**Q[state][action] **+** self**.**lr **\*** \

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

*# Следующее состояние считаем текущим*

state **=** next\_state

*# Суммарная награда за эпизод*

tot\_rew **+=** rew

**if** (done **or** truncated):

self**.**episodes\_reward**.**append(tot\_rew)

*# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Двойное Q-обучение \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

**class** DoubleQLearning\_Agent(BasicAgent):

'''

Реализация алгоритма Double Q-Learning

'''

*# Наименование алгоритма*

ALGO\_NAME **=** 'Двойное Q-обучение'

**def** \_\_init\_\_(self, env, eps**=**0.4, lr**=**0.1, gamma**=**0.98, num\_episodes**=**20000):

*# Вызов конструктора верхнего уровня*

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

*# Вторая матрица*

self**.**Q2 **=** np**.**zeros((self**.**nS, self**.**nA))

*# Learning rate*

self**.**lr**=**lr

*# Коэффициент дисконтирования*

self**.**gamma **=** gamma

*# Количество эпизодов*

self**.**num\_episodes**=**num\_episodes

*# Постепенное уменьшение eps*

self**.**eps\_decay**=**0.00005

self**.**eps\_threshold**=**0.01

**def** greedy(self, state):

'''

<<Жадное>> текущее действие

Возвращает действие, соответствующее максимальному Q-значению

для состояния state

'''

temp\_q **=** self**.**Q[state] **+** self**.**Q2[state]

**return** np**.**argmax(temp\_q)

**def** print\_q(self):

print('Вывод Q-матриц для алгоритма ', self**.**ALGO\_NAME)

print('Q1')

print(self**.**Q)

print('Q2')

print(self**.**Q2)

**def** learn(self):

'''

Обучение на основе алгоритма Double Q-Learning

'''

self**.**episodes\_reward **=** []

*# Цикл по эпизодам*

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

*# Начальное состояние среды*

state **=** self**.**get\_state(self**.**env**.**reset())

*# Флаг штатного завершения эпизода*

done **=** **False**

*# Флаг нештатного завершения эпизода*

truncated **=** **False**

*# Суммарная награда по эпизоду*

tot\_rew **=** 0

*# По мере заполнения Q-матрицы уменьшаем вероятность случайного выбора действия*

**if** self**.**eps **>** self**.**eps\_threshold:

self**.**eps **-=** self**.**eps\_decay

*# Проигрывание одного эпизода до финального состояния*

**while** **not** (done **or** truncated):

*# Выбор действия*

*# В SARSA следующее действие выбиралось после шага в среде*

action **=** self**.**make\_action(state)

*# Выполняем шаг в среде*

next\_state, rew, done, truncated, \_ **=** self**.**env**.**step(action)

**if** np**.**random**.**rand() **<** 0.5:

*# Обновление первой таблицы*

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**:

*# Обновление второй таблицы*

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])

*# Следующее состояние считаем текущим*

state **=** next\_state

*# Суммарная награда за эпизод*

tot\_rew **+=** rew

**if** (done **or** truncated):

self**.**episodes\_reward**.**append(tot\_rew)

**def** play\_agent(agent):

'''

Проигрывание сессии для обученного агента

'''

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**

env2**.**close()

**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)

**def** main():

*# run\_sarsa()*

*# run\_q\_learning()*

run\_double\_q\_learning()

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

main()

4. экранные формы с примерами выполнения программы.

0%| | 0/20000 [00:00<?, ?it/s]/home/ubuntu/miniforge3/envs/mmo/lib/python3.9/site-packages/gym/utils/passive\_env\_checker.py:233: DeprecationWarning: `np.bool8` is a deprecated alias for `np.bool\_`. (Deprecated NumPy 1.24)

if not isinstance(terminated, (bool, np.bool8)):

100%|██████████| 20000/20000 [00:04<00:00, 4124.78it/s]

Вывод Q-матриц для алгоритма Двойное Q-обучение

Q1

[[ 0. 0. 0. 0. 0.

0. ]

[ 0.50797274 4.55391946 -2.13448742 2.85151387 8.36234335

-5.31809942]

[ 5.03490358 3.90431984 1.60466939 5.09970705 13.27445578

-0.76129146]

...

[ -0.92606349 5.92867152 -1.20865181 -1.96339338 -4.90047816

-5.45333677]

[ -5.25788508 -5.33989175 -5.06078195 5.76431854 -9.28262056

-11.63296409]

[ 1.39059848 -0.1 1.65440971 17.61361317 0.

-2.97818268]]

Q2

[[ 0. 0. 0. 0. 0. 0. ]

[ 2.02889628 0.02249159 -1.42773754 -0.06443524 8.36234335 -4.25731818]

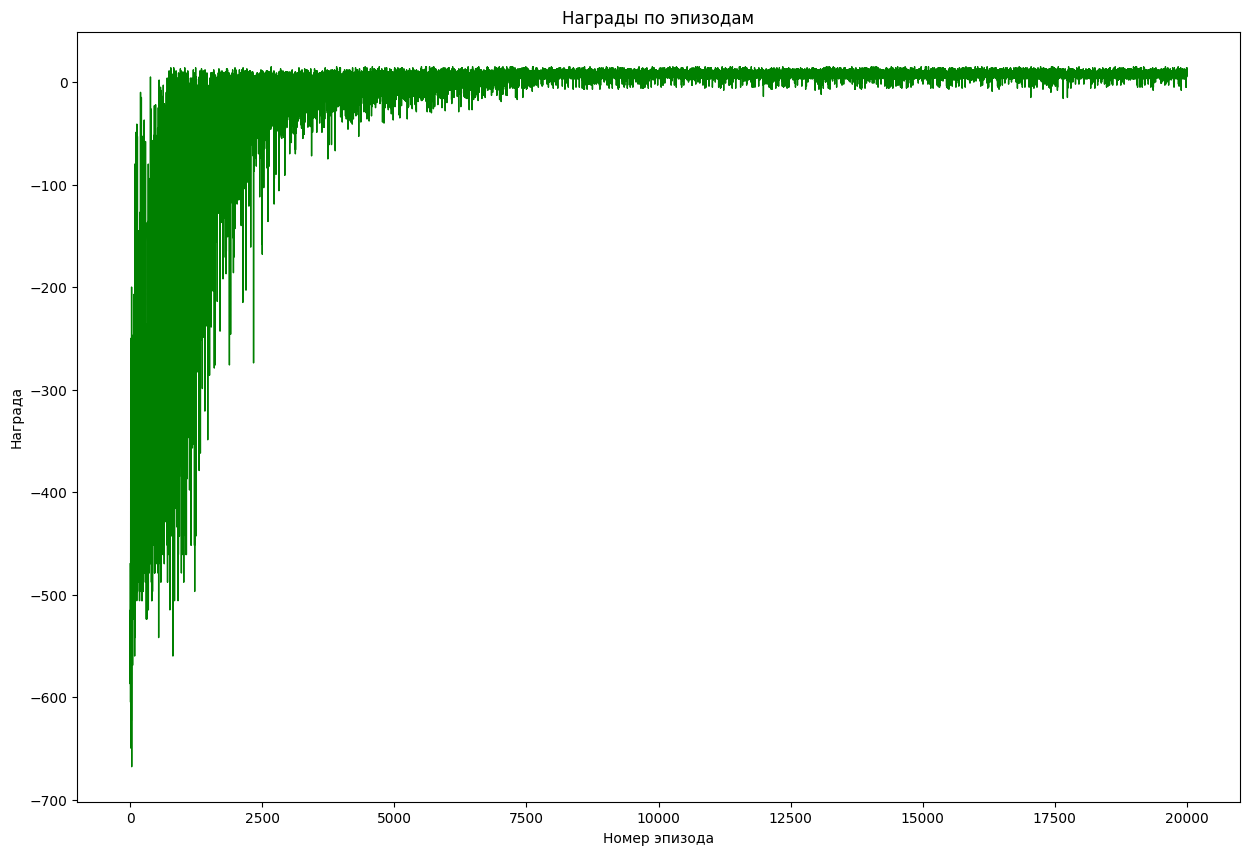
[ 7.07260023 6.09698691 3.28731399 7.99054284 13.27445578 1.13086475]

...

[-1.48767668 11.79867691 -2.05520107 0.84629559 -3.51257305 -3.51829407]

[-4.46434165 -4.99529228 -4.90322496 4.86655229 -9.05118778 -9.76693804]

[ 2.24642471 3.40995822 -0.1098 18.12243688 -0.39405902 -1.19770241]]

****

**Список литературы**

[1] Гапанюк Ю. Е. COURSE\_MMO\_SPRING\_2024// GitHub. –– 2024. –– Режим доступа: https://github.com/ugapanyuk/courses\_current/wiki/COURSE\_MMO\_SPRING\_2024

[2] <https://www.kaggle.com/datasets>