Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика, искусственный интеллект и системы управления»

Кафедра «Системы обработки информации и управления»

Отчет по рубежному контролю №2

по дисциплине «Методы машинного обучения»

Вариант 7

ИСПОЛНИТЕЛИ:

Пенегина В.В.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

Москва, 2023

Задание:

Для одного из алгоритмов временных различий, реализованных Вами в соответствующей лабораторная работе: SARSA, Q-обучение, ДвойноеQ-обучение.

Осуществите подбор гиперпараметров. Критерием оптимизации должна являться суммарная награда.

```
import numpy as np
import matplotlib.pyplot as plt
import gym
from tqdm import tqdm

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 в 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


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
среде
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
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 * \
self.Q[next_state][next_action] - self.Q[state][action])
# Правило обновления для Q-обучения
* \ self.Q[state][action])
# (rew + self.gamma *
self.Q[state][action] = self.Q[state][action] + self.lr(rew + self.gamma *
np.max(self.Q[next_state]) -

                                                    # 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('CliffWalking-v0', 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_q_learning():
    env = gym.make("CliffWalking-v0")


epsilons = [0.3, 0.4, 0.5]
learning_rates = [0.05, 0.1, 0.2]
gammas = [0.95, 0.98, 0.99]
num_episodes = 20000
best_reward = float('-inf')
best_hyperparams = None
best_agent = None
for eps in epsilons:
```

```
    for lr in learning_rates:
for gamma in gammas:
    agent = QLearning_Agent(env, eps=eps, lr=lr,
                            gamma=gamma, num_episodes=num_episodes)

agent.learn()
total_reward = sum(agent.episodes_reward)
print(f'Гиперпараметры: epsilon={eps}, learning rate={lr}, gamma={gamma}, num
episodes={num_episodes}')
print(f'Суммарная награда: {total_reward}\n')
if total_reward > best_reward:
    best_reward = total_reward
best_hyperparams = (eps, lr, gamma, num_episodes)
best_agent = agent
print(
    f'Лучшие гиперпараметры: epsilon={best_hyperparams[0]}, learning
rate={best_hyperparams[1]}, gamma={best_hyperparams[2]}, num
episodes={best_hyperparams[3]}')
print(f'Суммарная награда: {best_reward}\n')
best_agent.print_q()
best_agent.draw_episodes_reward()
play_agent(best_agent)


def main():
    run_q_learning()


if name == ' main ': main()
```

Результаты изменения гипермараметров:

3.1. Гиперпараметры: epsilon = 0.3, learning rate = 0.05, gamma = 0.95, num episodes = 20000

Суммарная награда: -734106

3.2. Гиперпараметры: epsilon = 0.3, learning rate = 0.05, gamma = 0.98, num episodes = 20000

Суммарная награда: -719783

3.3. Гиперпараметры: epsilon = 0.3, learning rate = 0.05, gamma = 0.99, num episodes = 20000

Суммарная награда: -724730

3.4. Гиперпараметры: epsilon = 0.3, learning rate = 0.1, gamma = 0.95, num episodes = 20000

Суммарная награда: -738206

3.5. Гиперпараметры: epsilon = 0.3, learning rate = 0.1, gamma = 0.98, num episodes = 20000

Суммарная награда: -723993

3.6. Гиперпараметры: epsilon = 0.3, learning rate = 0.1, gamma = 0.99, num episodes = 20000

Суммарная награда: -747135

3.7. Гиперпараметры: epsilon = 0.3, learning rate = 0.2, gamma = 0.95, num episodes = 20000

Суммарная награда: -690261

3.8. Гиперпараметры: epsilon = 0.3, learning rate = 0.2, gamma = 0.98, num episodes = 20000

Суммарная награда: -695174

3.9. Гиперпараметры: epsilon = 0.3, learning rate = 0.2, gamma = 0.99, num episodes = 20000

Суммарная награда: -697318

3.10. Гиперпараметры: epsilon = 0.4, learning rate = 0.05, gamma = 0.95, num episodes = 20000

Суммарная награда: -1187325

3.11. Гиперпараметры: epsilon = 0.4, learning rate = 0.05, gamma = 0.98, num episodes = 20000

Суммарная награда: -1187912

3.12. Гиперпараметры: epsilon = 0.4, learning rate = 0.05, gamma = 0.99, num episodes = 20000

Суммарная награда: -1187358

3.13. Гиперпараметры: epsilon = 0.4, learning rate = 0.1, gamma = 0.95, num episodes = 20000

Суммарная награда: -1067977

3.14. Гиперпараметры: epsilon = 0.4, learning rate = 0.1, gamma = 0.98, num episodes = 20000

Суммарная награда: -1073649

3.15. Гиперпараметры: epsilon = 0.4, learning rate = 0.1, gamma = 0.99, num episodes = 20000

4. Вывод

Лучшие гиперпараметры: epsilon = 0.3, learning rate = 0.2, gamma = 0.95, num episodes = 20000
Суммарная награда: -690261