****Московский государственный технический университет им. Н.Э. Баумана

Кафедра «Системы обработки информации и управления»

Лабораторная работа №4

по дисциплине

«Методы машинного обучения»

на тему

# **«Реализация алгоритма Policy Iteration»**

Выполнил:

студент группы ИУ5-22М

Джин Шуо

Москва — 2024 г.

**1. Цель лабораторной работы**

Oзнакомление с базовыми методами обучения с подкреплением.

**2. Задание**

На основе рассмотренного на лекции примера реализуйте алгоритм Policy Iteration для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).

**3. текст программы**

import gym

import numpy as np

import matplotlib.pyplot as plt

from pprint import pprint

class PolicyIterationAgent:

'''Класс, эмулирующий работу агента'''

def \_\_init\_\_(self, env):

self.env = env

# Пространство состояний

self.observation\_dim = env.observation\_space.n

# Массив действий в соответствии с документацией

self.actions\_variants = np.arange(env.action\_space.n)

# Задание стратегии (политики)

self.policy\_probs = np.full((self.observation\_dim, len(self.actions\_variants)), 1 / len(self.actions\_variants))

# Начальные значения для v(s)

self.state\_values = np.zeros(shape=(self.observation\_dim))

# Начальные значения параметров

self.maxNumberOfIterations = 1000

self.theta = 1e-6

self.gamma = 0.99

def print\_policy(self):

'''Вывод матриц стратегии'''

print('Стратегия:')

pprint(self.policy\_probs)

def policy\_evaluation(self):

'''Оценивание стратегии'''

# Предыдущее значение функции ценности

valueFunctionVector = self.state\_values

for iterations in range(self.maxNumberOfIterations):

# Новое значение функции ценности

valueFunctionVectorNextIteration = np.zeros(shape=(self.observation\_dim))

# Цикл по состояниям

for state in range(self.observation\_dim):

# Вероятности действий

action\_probabilities = self.policy\_probs[state]

# Цикл по действиям

outerSum = 0

for action, prob in enumerate(action\_probabilities):

innerSum = 0

# Цикл по вероятностям действий

for probability, next\_state, reward, isTerminalState in self.env.P[state][action]:

innerSum += probability \* (reward + self.gamma \* self.state\_values[next\_state])

outerSum += self.policy\_probs[state][action] \* innerSum

valueFunctionVectorNextIteration[state] = outerSum

if np.max(np.abs(valueFunctionVectorNextIteration - valueFunctionVector)) < self.theta:

# Проверка сходимости алгоритма

valueFunctionVector = valueFunctionVectorNextIteration

break

valueFunctionVector = valueFunctionVectorNextIteration

return valueFunctionVector

def policy\_improvement(self):

'''Улучшение стратегии'''

qvaluesMatrix = np.zeros((self.observation\_dim, len(self.actions\_variants)))

improvedPolicy = np.zeros((self.observation\_dim, len(self.actions\_variants)))

# Цикл по состояниям

for state in range(self.observation\_dim):

for action in range(len(self.actions\_variants)):

for probability, next\_state, reward, isTerminalState in self.env.P[state][action]:

qvaluesMatrix[state, action] += probability \* (reward + self.gamma \* self.state\_values[next\_state])

# Находим лучшие индексы

bestActionIndex = np.where(qvaluesMatrix[state, :] == np.max(qvaluesMatrix[state, :]))

# Обновление стратегии

improvedPolicy[state, bestActionIndex] = 1 / np.size(bestActionIndex)

return improvedPolicy

def policy\_iteration(self, cnt):

'''Основная реализация алгоритма'''

policy\_stable = False

for i in range(1, cnt + 1):

self.state\_values = self.policy\_evaluation()

self.policy\_probs = self.policy\_improvement()

print(f'Алгоритм выполнился за {i} шагов.')

def play\_agent(agent):

env2 = gym.make('CliffWalking-v0', render\_mode='human')

state = env2.reset()[0]

done = False

while not done:

p = agent.policy\_probs[state]

if isinstance(p, np.ndarray):

action = np.random.choice(len(agent.actions\_variants), p=p)

else:

action = p

next\_state, reward, terminated, truncated, \_ = env2.step(action)

env2.render()

state = next\_state

if terminated or truncated:

done = True

def main():

# Создание среды

env = gym.make('CliffWalking-v0')

env.reset()

# Обучение агента

agent = PolicyIterationAgent(env)

agent.print\_policy()

agent.policy\_iteration(1000)

agent.print\_policy()

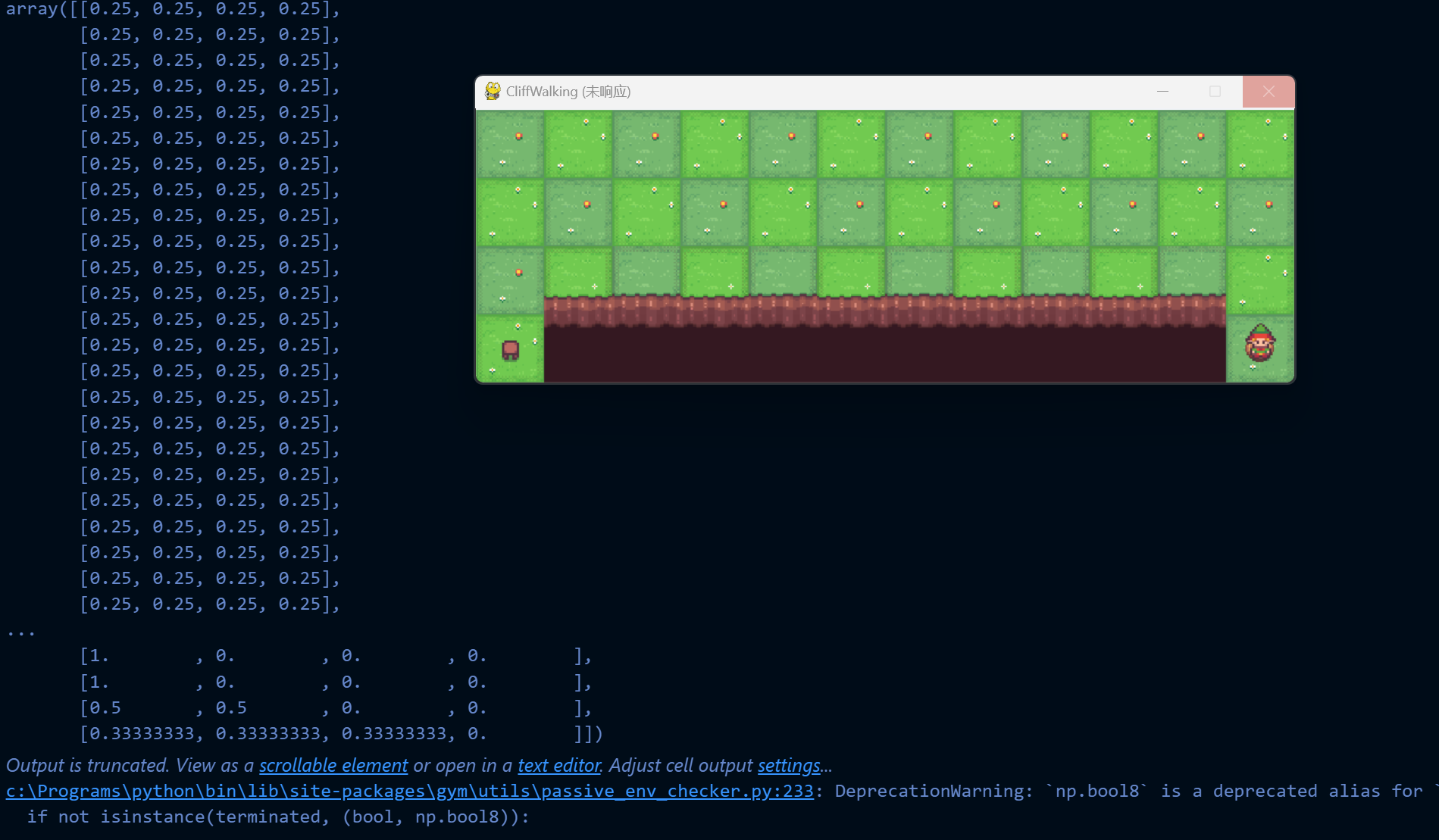
# Проигрывание сцены для обученного агента

play\_agent(agent)

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

main()

1. **экранные формы с примерами выполнения программы.**



**Список литературы**

[1] https://github.com/ugapanyuk/courses\_current/wiki/LAB\_MMO\_policy\_iteration