**МІНІСТЕРСТВО ОСВІТИ І НАУКИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Інститут комп'ютерних наук та інформаційних технологій**

**Кафедра систем штучного інтелекту**



**Звіт до лабораторної роботи №4**

**з дисципліни “ Машинне навчання ”**

**На тему: “ Навчання з підкріпленням”**

**Виконав**:

ст. гр. КН-311

Ткачук Орест

**Викладач:**

Бойчук А. Р.

Львів – 2021

**Мета:** засвоїти основні відомості при роботі з навчанням з підкріпленням. Навчитись використовувати його на практиці.

**Завдання:** реалізувати алгоритм Q-learning для вирішення CartPole середовища OpenAi-gym

1. Ознайомитись з OpenAI gym документацією <https://gym.openai.com/docs/>
2. Ознайомитись з віртуальним середовищем CartPole-v1
3. Реалізувати алгоритм Q-learning та навчити агента балансувати карт
4. Зобразити у вигляді відео результат - балансуючу cart pole понад 100 фреймів
5. Результат тренування зберегти у файл для можливості відтворення гри під час здачі лабораторної роботи. Ці ж артефакти завантажити на внс.

**Хід роботи**

1. Ознайомитись з OpenAI gym документацією <https://gym.openai.com/docs/>
2. Ознайомитись з віртуальним середовищем CartPole-v1

import gym

import time

env = gym.make('CartPole-v1')

env.reset()

for \_ in range(1000):

env.render()

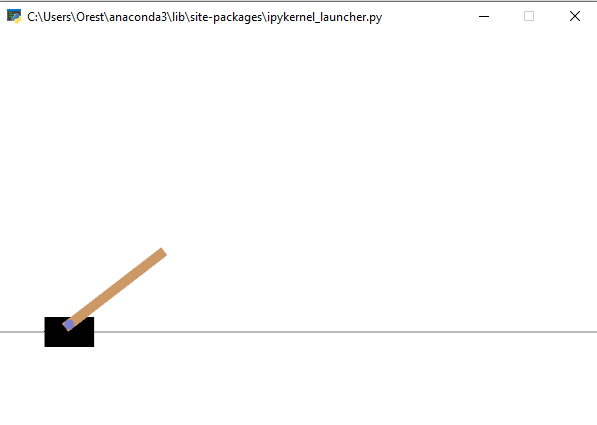
env.step(0)

time.sleep(0.02)

env.step(1)

time.sleep(0.02)

env.close()



1. Реалізувати алгоритм Q-learning та навчити агента балансувати карт

Для початку завантажимо середовище CartPole-v1 і визначимо, скільки просторів станів та дій має це середовище

env = gym.make('CartPole-v1')

n\_actions = env.action\_space.n

n\_states = env.observation\_space.shape[0]

print("Action space size: ", n\_actions)

print("State space size: ", n\_states)



Далі перевіримо максимальні та мінімальні значення станів

print('\nStates high values:')

print(env.observation\_space.high[0])

print(env.observation\_space.high[1])

print(env.observation\_space.high[2])

print(env.observation\_space.high[3])

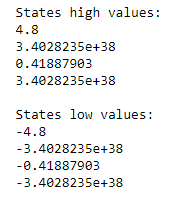
print('\nStates low values:')

print(env.observation\_space.low[0])

print(env.observation\_space.low[1])

print(env.observation\_space.low[2])

print(env.observation\_space.low[3])



Значення 2 та 4 станів є безкінечні, тому їх потрібно буде дискретизувати. Для цього визначимо межі

upper\_bounds = [

env.observation\_space.high[0],

0.5,

env.observation\_space.high[2],

math.radians(50)

]

lower\_bounds = [

env.observation\_space.low[0],

-0.5,

env.observation\_space.low[2],

-math.radians(50)

]

Створимо пустий Q-table

Q = np.zeros(buckets + (n\_actions,))

print(np.shape(Q))



Визначимо алгоритм ϵ-greedy exploration strategy, в якому му будемо отримувати випадкове число і в залежності того чи воно більше, чи менше від eps робити одну з двох дій - exploitation або exploration

def epsilon\_greedy\_strategy(state, eps):

random\_exploration\_exploitation = np.random.random()

if random\_exploration\_exploitation <= eps:

new\_action = env.action\_space.sample()

else:

new\_action = np.argmax(Q[state])

return new\_action

Реалізуємо train – цикл

rewards\_for\_episodes = []

for episode in range(n\_episodes):

current\_state = env.reset()

current\_state = discretize(current\_state)

alpha = max(min\_alpha, min(1.0, 1.0 - math.log10((episode + 1) / decay\_rate)))

epsilon = max(min\_epsilon, min(1.0, 1.0 - math.log10((episode + 1) / decay\_rate)))

episode\_reward\_acc = 0

for t in range(n\_steps):

#env.render()

action = epsilon\_greedy\_strategy(current\_state, epsilon)

new\_state, reward, done, \_ = env.step(action)

new\_state = discretize(new\_state)

update\_q\_table(current\_state, action, reward, new\_state, alpha)

current\_state = new\_state

episode\_reward\_acc += reward

if done:

print('Episode:{}/{} failed with a total reward of: {}'.format(episode, n\_episodes, episode\_reward\_acc))

break

#if episode\_reward\_acc == 200:

#break

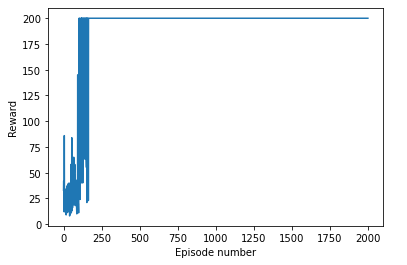
rewards\_for\_episodes.append(episode\_reward\_acc)

Для кожного епізоду будемо проводити n кроків. Для цього визначаємо поточний стан, вирішуємо який з кроків greedy exploration strategy буде виконуватися, дискретизуємо нескінченні значення стану, оновлюємо Q-table і змінюємо поточний стан. Оновлення Q-table реалізовано наступною функцією

def update\_q\_table(current\_state, action, reward, new\_state, alpha):

Q[current\_state][action] += alpha \* (reward + gamma \* np.max(Q[new\_state]) - Q[current\_state][action])

Значення винагород для кожного епізоду



1. Зобразити у вигляді відео результат - балансуючу cart pole понад 100 фреймів

Створимо тестовий цикл в якому будемо використовувати уже оновлений Q-table, та будемо рендерити середовище протягом 200 кроків(200 кадрів)

Q = np.load('Q\_table.npy')

current\_state = env.reset()

current\_state = discretize(current\_state)

episode\_reward = 0

test\_steps = 200

for t in range(test\_steps):

env.render()

action = greedy\_strategy(current\_state)

new\_state, reward, done, \_ = env.step(action)

new\_state = discretize(new\_state)

update\_q\_table(current\_state, action, reward, new\_state, min\_alpha)

current\_state = new\_state

episode\_reward += reward

time.sleep(0.02)

if done:

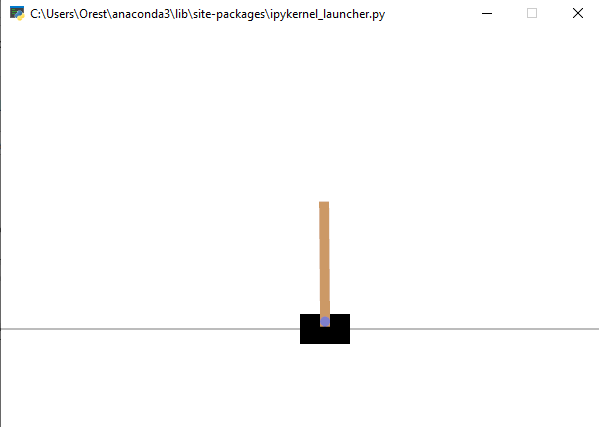
print('Test episode failed with a total reward of: {}'.format(episode\_reward))

break

print('Test episode(with {} steps) finished with a total reward of: {}'.format(test\_steps, episode\_reward))

env.close()

Результат виконання



1. Результат тренування зберегти у файл для можливості відтворення гри під час здачі лабораторної роботи. Ці ж артефакти завантажити на внс

np.save('Q\_table.npy', Q)