**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **ІКНІ**

Кафедра **СШІ**

**ЗВІТ**

До лабораторної роботи №4

**З дисципліни:**

**Машинне Навчання**

**Виконав:**

ст. гр. КН-308

Матвіїв М.А.

**Прийняв:**

Якимишин Х.М.

Львів – 2020

Тема:​ навчанням з підкріпленням.

Мета: засвоїти основні відомості про роботу з навчанням з підкріпленням.  
Навчитись використовувати його на практиці.

Основні відомостіOpenAI Gym це набір утиліт для розробки та порівняння алгоритмів  
навчання з підкріпленням(Reinforcement learning algorithms). Він підтримує навчання агентів різноманітним завданням - від ходьби до гри в такі ігри як теніс або пейнбол.  
Нижче можна переглянути простий приклад коду, необхідний для того  
щоб запустити щось в OpenAI Gym. Він запустить об’єкт середовища  
CartPole-v0 тривалістю 1000 кроків, при цьому генеруючи середовище для  
кожного з цих кроків. Ви повинні побачити поп-ап вікно з рендером.

**Q-learning**

Q-learning це алгоритм навчання, який вивчає якість чи цінність пари стан/дія. Відповідно, таким чином дає відповідь на питання, яка дія найкраща в поточному стані.

Qhat(st,at) = (1-ɑ)Qhat(st,at) + ɑ(r + ƔmaxaQ(s′, a′))

Кінцевим результатом цього алгоритму буде Q-table - таблиця розмірністю size\_of\_state\_space \* size\_of\_action\_space. Відповідно для того щоб дізнатись яку дію обрати в кожному з можливих станів достатньо обрати дію яка має найбільше Q-value в цьому стані.

**ϵ-greedy exploration strategy**

Для ефективного навчання необхідно також пам’ятати про exploitation-exploration trade-off. Іншими словами, скільки з вже вивченої інформації ми використовуємо, а скільки пробуємо досліджувати всесвіт, виконуючи не завжди оптимальні на нашу думку дії.

Однією з таких стратегій є ϵ-greedy стратегія, яка має насправді дуже просту ідею. Ε мале число, яке вказує на ймовірність того, що на кожному кроці буде виконана якась випадкова дія, замість тої що вважається оптимальною зараз.

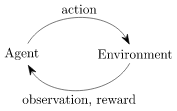
**Спостереження**

Якщо ми хочемо мати агента, який не просто робить випадкові дії на кожному кроці, буде хорошою ідеєю знати як власне наші дії впливають на середовище.

Функція середовища step повертає власне те що нам потрібно. Вона повертає 4 значення:

* observation (object): специфічний до середовища об’єкт який відображає спостережуваний агентом стан середовища. Наприклад, пікселі з камери, кути та швидкості робота, або стан ігрової дошки в настільних іграх.
* reward (float): розмір винагороди яку ми отримали за виконання попередньої дії. Шкала винагороди різна для кожного середовища, однак ціль завжди однакова - збільшити отримувану винагороду.
* done (boolean): прапорець, що вказує чи вже потрібно викликати reset для перевантаження середовища. Більшість(але не всі) завдання розділені на добре описані епізоди, відповідно done буде True якщо епізод завершений.
* info (dict): діагностична інформація що корисна для відлагодження. Ця інформація інколи буде корисна для навчання (наприклад, вона може містити ймовірності, які стоять за останньою зміною середовища). Однак агенти для порівняння на офіційному лідерборді не можуть використовувати цю інформацію.

Це приклад реалізації класичного циклу “агент-середовище”.



## 

**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 карти
4. Зобразити у вигляді відео результат - балансуючу cart pole понад 100 епізодів

**Код Програми**

from sklearn.preprocessing import KBinsDiscretizer

import numpy as np

import time, math, random

from typing import Tuple

import gym

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

policy = lambda obs: 1

for \_ in range(5):

obs = env.reset()

for \_ in range(80):

actions = policy([\*obs])

obs, reward, done, info = env.step(actions)

env.render()

time.sleep(0.05)

env.close()

n\_bins = ( 6 , 12 )

lower\_bounds = [ env.observation\_space.low[2], -math.radians(50) ]

upper\_bounds = [ env.observation\_space.high[2], math.radians(50) ]

def discretizer( \_ , \_\_ , angle, pole\_velocity ) -> Tuple[int,...]:

"""Convert continues state intro a discrete state"""

est = KBinsDiscretizer(n\_bins=n\_bins, encode='ordinal', strategy='uniform')

est.fit([lower\_bounds, upper\_bounds ])

return tuple(map(int,est.transform([[angle, pole\_velocity]])[0]))

Q\_table = np.zeros(n\_bins + (env.action\_space.n,))

Q\_table.shape

def policy( state : tuple ):

"""Choosing action based on epsilon-greedy policy"""

return np.argmax(Q\_table[state])

def new\_Q\_value( reward : float , new\_state : tuple , discount\_factor=1 ) -> float:

"""Temperal diffrence for updating Q-value of state-action pair"""

future\_optimal\_value = np.max(Q\_table[new\_state])

learned\_value = reward + discount\_factor \* future\_optimal\_value

return learned\_value

# Adaptive learning of Learning Rate

def learning\_rate(n : int , min\_rate=0.01 ) -> float :

"""Decaying learning rate"""

return max(min\_rate, min(1.0, 1.0 - math.log10((n + 1) / 25)))

def exploration\_rate(n : int, min\_rate= 0.1 ) -> float :

"""Decaying exploration rate"""

return max(min\_rate, min(1, 1.0 - math.log10((n + 1) / 25)))

n\_episodes = 300

for e in range(n\_episodes):

# Siscretize state into buckets

current\_state, done = discretizer(\*env.reset()), False

while done==False:

# policy action

action = policy(current\_state) # exploit

# insert random action

if np.random.random() < exploration\_rate(e) :

action = env.action\_space.sample() # explore

# increment enviroment

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

new\_state = discretizer(\*obs)

# Update Q-Table

lr = learning\_rate(e)

learnt\_value = new\_Q\_value(reward , new\_state )

old\_value = Q\_table[current\_state][action]

Q\_table[current\_state][action] = (1-lr)\*old\_value + lr\*learnt\_value

current\_state = new\_state

# Render the cartpole environment

env.render()

**Посилання на відео:** <https://youtu.be/REhvp8xvva0>

**Висновок**

На лабораторній роботі я ознайомився з алгоритмом Q-Learning, який являється динамічним навчальним алгоритмом.

В даній роботи ми навчили за допомогою Q-Learning алгоритму підставку балансувати так, щоб палиця, яка стоїть на ній, не падала як можна довший проміжок часу.