```
#!pip install -r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt

import gymnasium as gym
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
import random
```

☞We are using the `Taxi-v3` environment from OpenAI's gym: https://gym.openai.com/envs/Taxi-v3/

☞`Taxi-v3` is an easy environment because the action space is small, and the state space is large but finite.

☞Environments with a finite number of actions and states are called tabular

Actions

There are 6 discrete deterministic actions:

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pickup passenger
- 5: drop off passenger

Rewards

- -1 per step unless other reward is triggered.
- +20 delivering passenger.
- -10 executing "pickup" and "drop-off" actions illegally.

```
env=gym.make("Taxi-v3",  render_mode="rgb_array")

class RandomAgent:

    def __init__(self,env):
        self.env=env

    def get_Random_Action(self):
```

```
         # -----add the code------------
        return self.env.action_space.sample()
```

create a random Agent interacting with "Taxi-v3" environment

```
agent= RandomAgent(env)
```

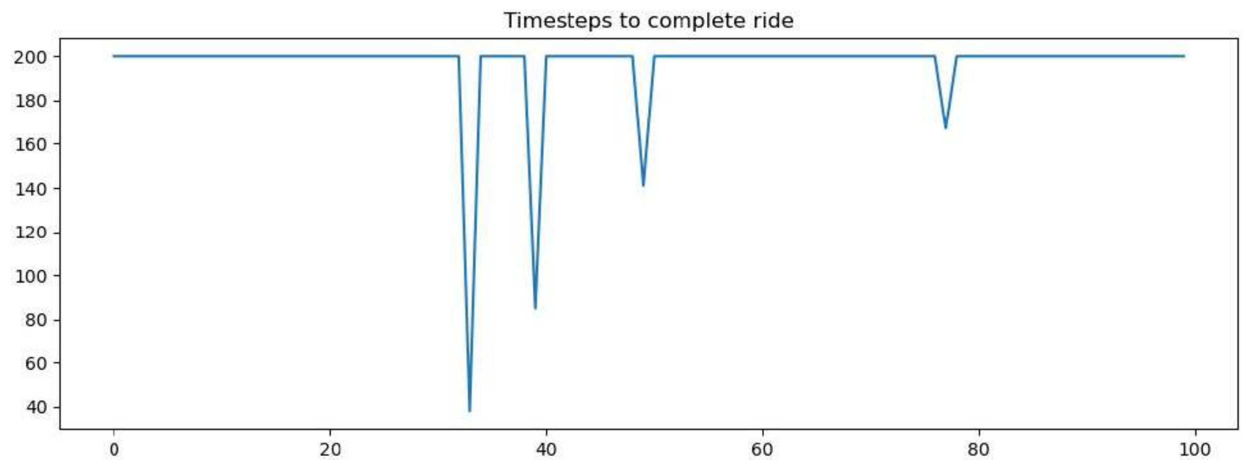
Train the Random agent on multiples episodes

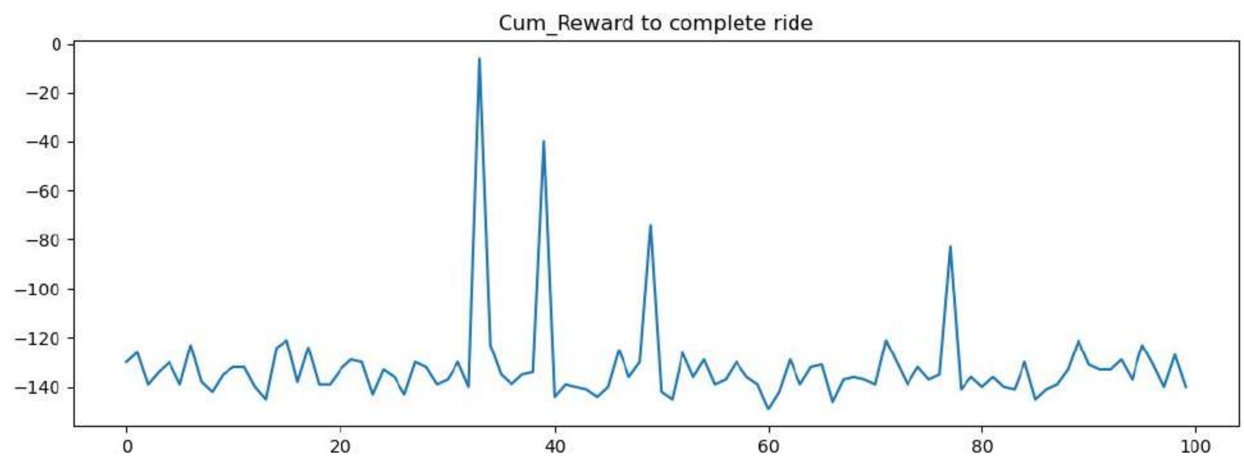
```

n_episodes = 100

# For plotting metrics
timesteps_per_episode = []
cumReward_per_episode = []
penalties_per_episode = []

for i in tqdm(range(0, n_episodes)):

    # reset environment to a random state
    #--- add code here----------
    env.reset()
    # initialize the metrics
    steps, penalties, cum_reward, = 0, 0, 0
    done = False

    while not done:

        # select an action randomly
        action=agent.get_Random_Action()

        #execute the selected action on the environment and return the
variables:
        #==>next_state, reward, terminated and truncated
        #--- add code here----------
        next_state, reward, terminated, truncated, _ =
senv.step(action)

        if reward == -10:
            penalties += 1
        else:
            cum_reward+=reward


        steps += 1

        done=terminated or truncated


    timesteps_per_episode.append(steps)
```

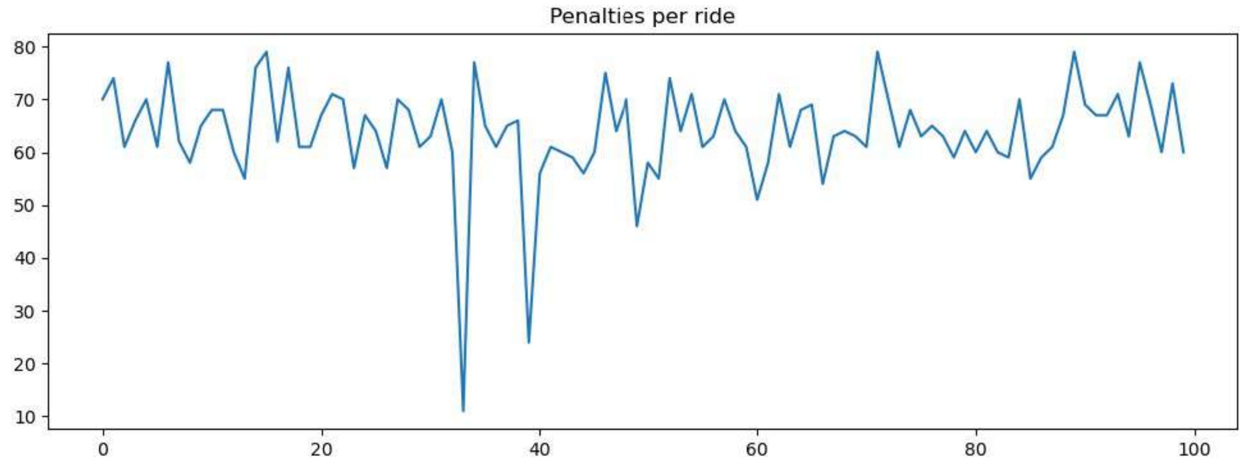
```
    cumReward_per_episode.append(cum_reward)
    penalties_per_episode.append(penalties)
```

```
100%|██████████| 100/100 [00:00<00:00, 202.94it/s]
```

```
fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Timesteps to complete ride")
pd.Series(timesteps_per_episode).plot(kind="line")
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Cum_Reward to complete ride")
pd.Series( cumReward_per_episode).plot(kind="line")
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Penalties per ride")
pd.Series(penalties_per_episode).plot(kind="line")
plt.show()
```






Penalties per ride
80
70
60
50
40
30
20
10
0
20
40
60
80
100

5mn4lg6ox

December 17, 2023

[ ]:
```
#!pip install -r https://raw.githubusercontent.com/malkiAbdelhamid/
↪Advanced-Deep-Learning-2023-2024-esisba/master/lab1_QLearning/
↪requirements_lab1.txt
```

Now that we studied the Q-Learning algorithm, let's implement it from scratch and train our Q-Learning agent in Taxi-3 environment:

Q-Learning **is the RL algorithm that**:

- Trains *Q-Function*, an **action-value function** that encoded, in internal memory, by a *Q-table* **that contains all the state-action pair values.**
- Given a state and action, our Q-Function **will search the Q-table for the corresponding value.**
- When the training is done,**we have an optimal Q-Function, so an optimal Q-Table.**
- And if we **have an optimal Q-function**, we have an optimal policy, since we **know for, each state, the best action to take.**

0.0.1 Import the packages

[5]:
```
import gymnasium as gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from tqdm import tqdm
import imageio
```

0.0.2 QAgent Class:

Is initialized with the environment *env* and the necessary hyper-parameters (*alpha*, *gamma*, *epsilon*, etc)

This class implements the most important steps of **Q-Learning** Algorithm: - Q-Table initialization - Q-Table shape=(n_states * n_actions)

- Epsilon-greedy policy as an acting policy
  - With *probability 1-* : **we do exploitation** (i.e. our agent selects the action with the highest state-action pair value).

- With *probability* : we do **exploration** (trying a random action).

- Greedy-policy as an updating policy
  - using Bellman equation *Q(s,a) + lr[R(s,a) + gamma * max Q(s',a') - Q(s,a)]*

[19]:
```
class QAgent:

    def __init__(self, env, alpha, gamma, max_epsilon, min_epsilon, decay_rate):

        self.env = env

        # table with q-values: n_states * n_actions
        self.q_table = np.zeros(shape=(env.observation_space.n, env.
 ↪action_space.n))

        # hyper-parameters
        self.alpha = alpha                          # learning rate
        self.gamma = gamma                          # discount factor
        self.max_epsilon = max_epsilon              # Exploration probability at␣
 ↪start
        self.min_epsilon = min_epsilon              # Minimum exploration␣
 ↪probability
        self.decay_rate =  decay_rate               # Exponential decay rate for␣
 ↪exploration prob



    # get action using epsilon greedy policy
    def get_action(self, state, epsilon):
        # Randomly generate a number between 0 and 1
        random_num =random.random()

        # if random_num > greater than epsilon --> exploitation
        if random_num > epsilon:
            # Take the action with the highest value given a state
            action = np.argmax(self.q_table[state, :])
        # else --> exploration
        else:
            action = self.env.action_space.sample()

        return action

    def update_parameters(self, state, action, reward, next_state):
        """""""
        # Q-learning formula
        # Update Q(s,a):= Q(s,a) + lr [R(s,a)  + gamma * max Q(s',a') - Q(s,a)]
        old_value = self.q_table[state, action]
        next_max = np.max(self.q_table[next_state])
```

```
        new_value = old_value + self.alpha * (reward + self.gamma*next_max -␣
↪old_value)

        # update the q_table
        self.q_table[state, action] = new_value
    def sarsa_update_parameters(self, state, action, reward, next_state,␣
↪epsilon):
        """"""
        # Q-learning formula
        # Update Q(s,a):= Q(s,a) + lr [R(s,a)  + gamma * Q(s',a') - Q(s,a)]
        old_value = self.q_table[state, action]
        next_action = self.get_action(next_state, epsilon)
        next_value = self.q_table[next_state, next_action]
        new_value = old_value + self.alpha * (reward + self.gamma*next_value -␣
↪old_value)

        # update the q_table
        self.q_table[state, action] = new_value
        return next_action
```

0.0.3 The training loop:

The training is based on *Temporal Difference (TD) learning* where the Q-Table is updated after each step

- For episode in the total of training episodes:
  - Reduce epsilon (since we need less and less exploration)
  - Reset the environment
  - For step in max timesteps:
    * Choose the action a using epsilon greedy policy
    * Take the action (a) and observe the outcome state(s') and reward (r)
    * Update the Q-value Q(s,a) using Bellman equation Q(s,a) + lr [R(s,a) + gamma * max Q(s',a') - Q(s,a)]
    * If done, finish the episode
    * Our next state is the new state

[20]:
```
def train(n_training_episodes,  env, agent):

    episode_rewards = []
    episode_penalties = []
    episode_steps = []
    for episode in tqdm(range(n_training_episodes)):

        total_rewards_ep = 0
        total_penalties_ep=0
        total_steps_ep=0
```

```
        # Reduce epsilon (because we need less and less exploration)
        epsilon = agent.min_epsilon + (agent.max_epsilon - agent.
↪min_epsilon)*np.exp(-agent.decay_rate*episode)

        # Reset the environment
        state, info = env.reset()
        step = 0
        terminated = False
        truncated = False
        done=False
        action = agent.get_action(state, epsilon)
        # repeat
        while not done:
            # Choose the action At using epsilon greedy policy
            #action = agent.get_action(state, epsilon)

            # Take action At and observe Rt+1 and St+1
            # Take the action (a) and observe the outcome state(s') and reward␣
↪(r)
            next_state, reward, terminated, truncated, info = env.step(action)

            total_rewards_ep += reward
            total_steps_ep +=1
            if reward == -10:
                total_penalties_ep += 1

            # Update Q(s,a):= Q(s,a) + lr [R(s,a)  + gamma * max Q(s',a') -␣
↪Q(s,a)]
            #agent.sarsa_update_parameters(state, action, reward, next_state)
            next_action = agent.sarsa_update_parameters(state, action, reward,␣
↪next_state, epsilon)

            # If terminated or truncated finish the episode
            done=terminated or truncated

            if done:
                break

            # Our next state is the new state
            #--- add code here----------
            state = next_state
            action = next_action

        episode_rewards.append(total_rewards_ep)
        episode_steps.append(total_steps_ep)
        episode_penalties.append(total_penalties_ep)
```

```
    return  episode_rewards,episode_steps, episode_penalties
```

0.0.4 Hyper-Parameters

The exploration related hyper-paramters are some of the most important ones. - We need to make sure that our agent **explores enough of the state space** to learn a good value approximation. To do that, we need to have progressive decay of the epsilon. - If you decrease epsilon too fast (too high decay_rate), **you take the risk that your agent will be stuck**, since your agent didn't explore enough of the state space and hence can't solve the problem.

[24]:
```
# Training parameters
n_training_episodes = 10000  # Total training episodes
alpha= 0.7                   # Learning rate

# Environment parameters
gamma = 0.95                 # Discounting rate

# Exploration parameters
max_epsilon = 1.0             # Exploration probability at start
min_epsilon = 0.05            # Minimum exploration probability
decay_rate = 0.0005           # Exponential decay rate for exploration prob

# Evaluation parameters
n_eval_episodes = 100        # Total number of test episodes
```

0.0.5 Train the Q-Agent on Taxiv3 Environment

[25]:
```
env=gym.make("Taxi-v3", render_mode="rgb_array")

agent=QAgent(env, alpha, gamma, max_epsilon, min_epsilon, decay_rate)

episode_rewards,episode_steps, episode_penalties=train(n_training_episodes,␣
 ↪env,agent)
```

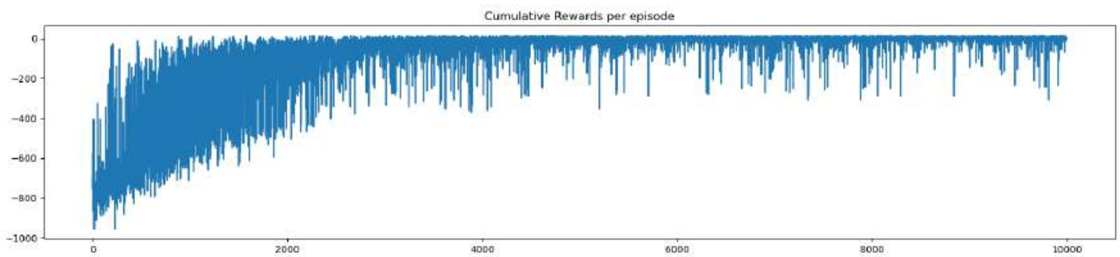
```
100%|      | 10000/10000 [00:11<00:00, 897.01it/s]
```

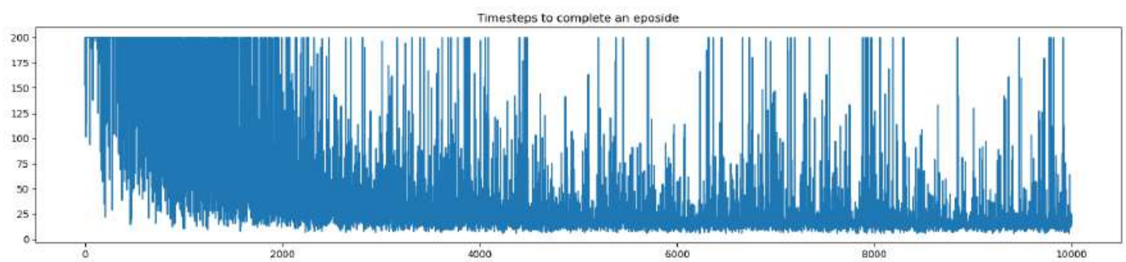
0.0.6 Plot the training result

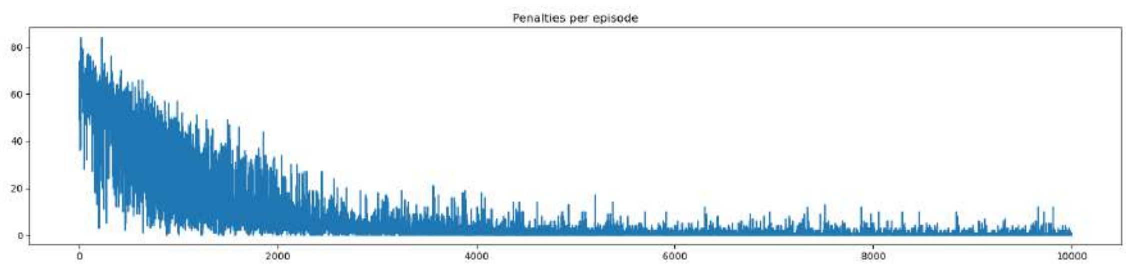
- episode_rewards: the cumulative rewards progression over all episodes (should be increased over time)
- episode_steps: the required step per episode (should be decreased over time)
- episode_penalties: the total penalties per episode (should be decreased over time)

1 SARSA 10000 ep

[26]:
```
fig, ax = plt.subplots(figsize = (20, 4))
ax.set_title("Cumulative Rewards per episode")
pd.Series(episode_rewards).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (20, 4))
ax.set_title("Timesteps to complete an eposide")
pd.Series(episode_steps).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (20, 4))
ax.set_title("Penalties per episode")
pd.Series(episode_penalties).plot(kind='line')
plt.show()
```

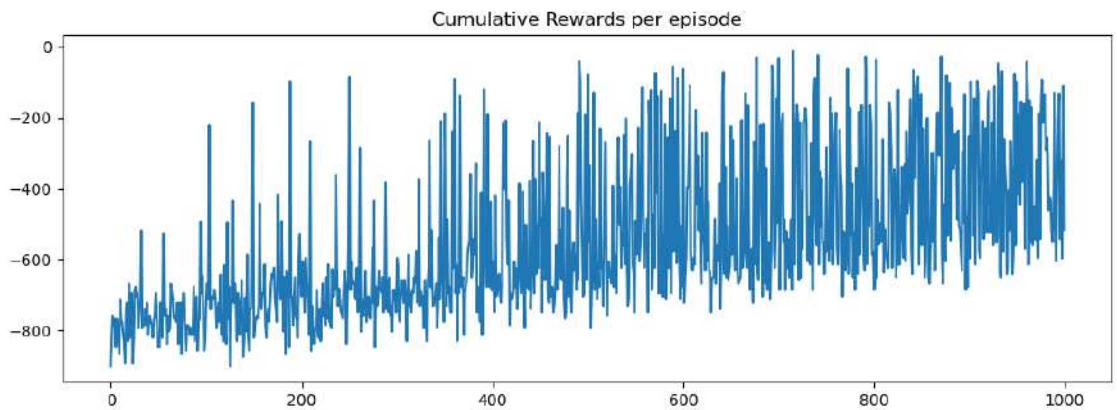


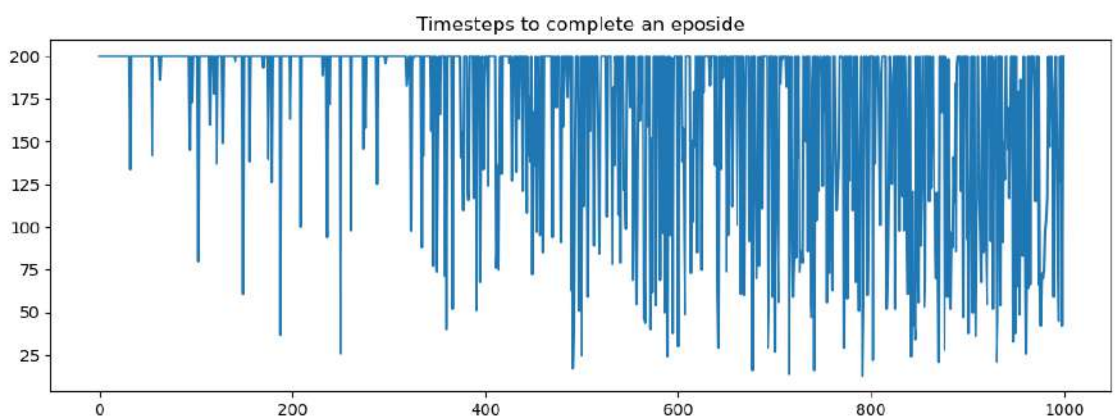


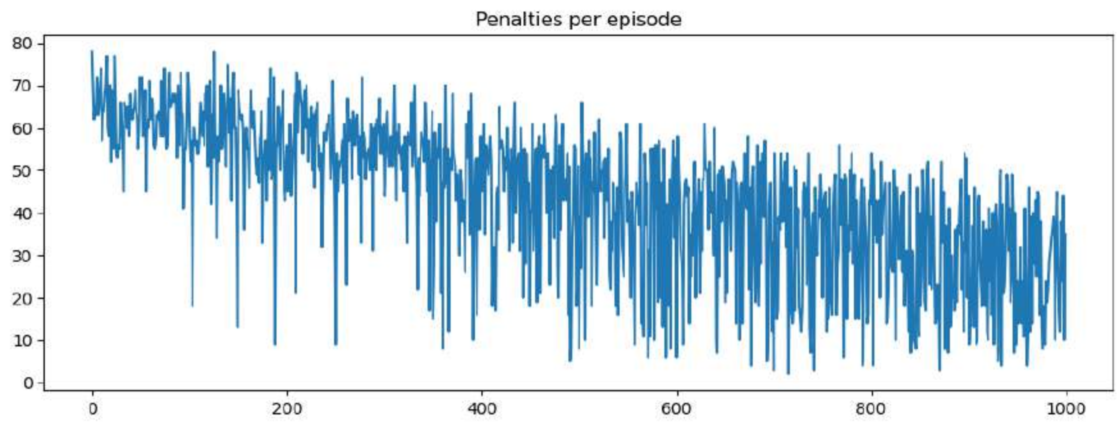


2 SARSA 1000 ep

[23]:
```
fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Cumulative Rewards per episode")
pd.Series(episode_rewards).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Timesteps to complete an eposide")
pd.Series(episode_steps).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Penalties per episode")
pd.Series(episode_penalties).plot(kind='line')
plt.show()
```

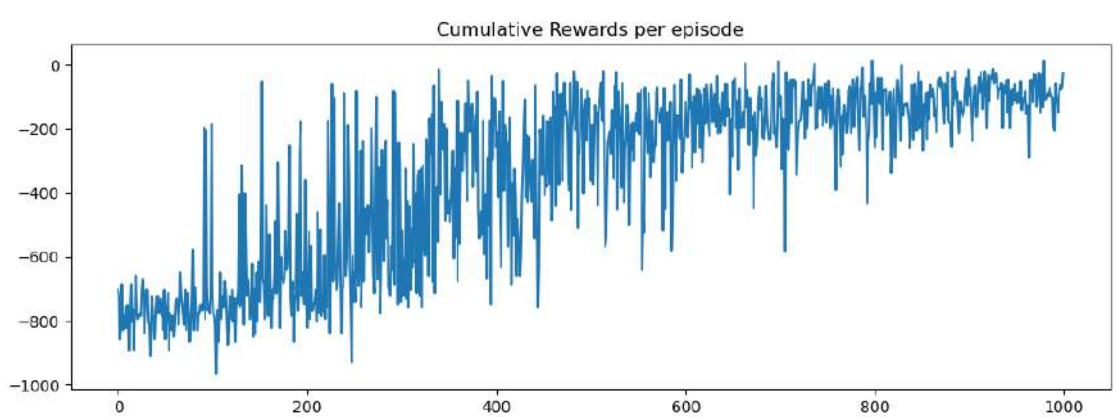


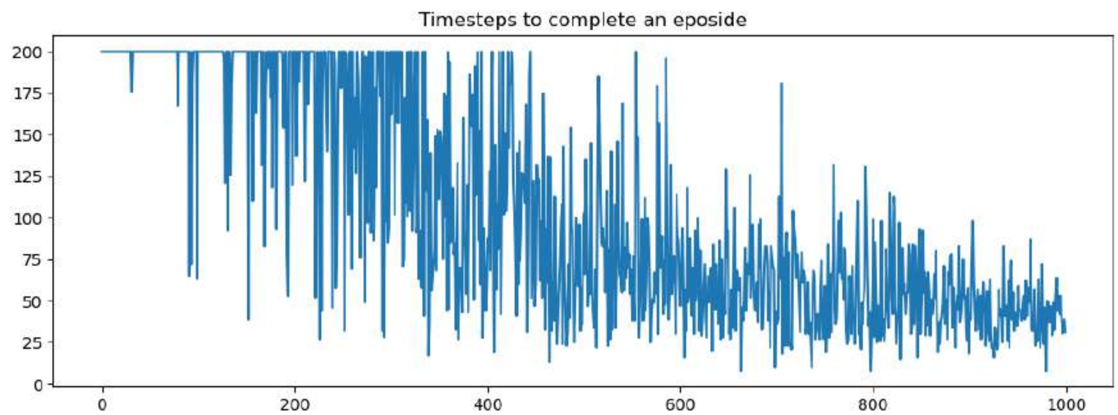


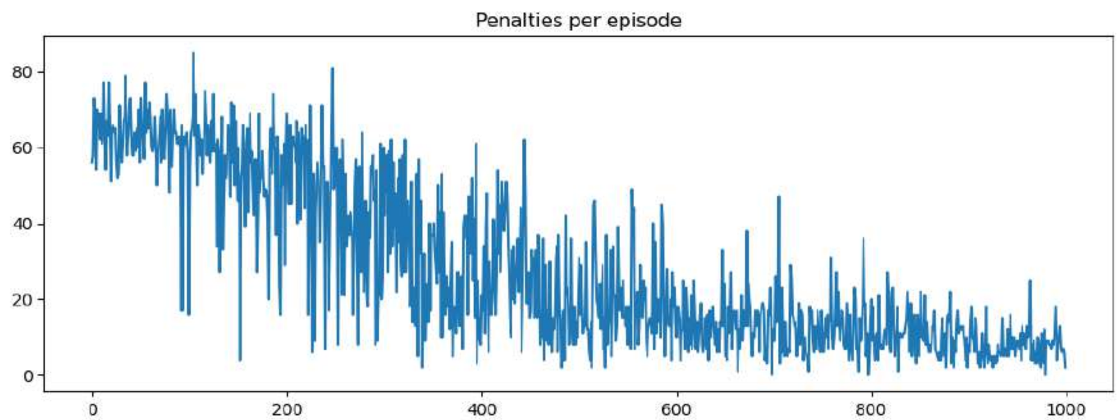


3 Q-Learning 1000 ep

[114]:
```
fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Cumulative Rewards per episode")
pd.Series(episode_rewards).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Timesteps to complete an eposide")
pd.Series(episode_steps).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Penalties per episode")
pd.Series(episode_penalties).plot(kind='line')
plt.show()
```







3.0.1 Evaluate the QAgent after training

[12]:
```
def evaluate_agent(env,agent,  n_eval_episodes):
    """
    Evaluate the agent for ``n_eval_episodes`` episodes and returns average␣
 ↪reward and std of reward.
    :param agent: the gent within its  evaluation environment and Qtable
    :param max_steps: Maximum number of steps per episode
    :param n_eval_episodes: Number of episode to evaluate the agent
    """
    episode_rewards = []
    episode_penalties = []
    episode_steps = []
    for episode in tqdm(range(n_eval_episodes)):

        state, info= env.reset()
        step = 0
```

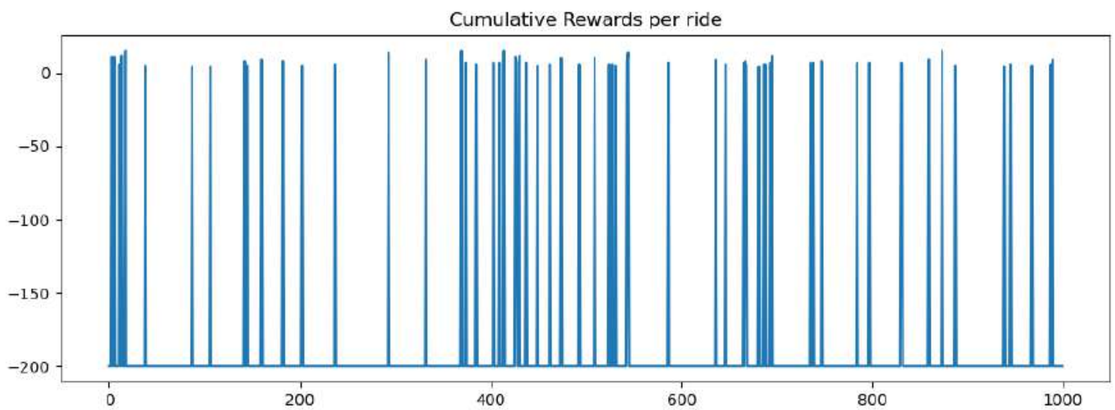
```
        truncated = False
        terminated = False
        total_rewards_ep = 0
        total_penalties_ep=0
        total_steps_ep=0
        done= False

        while not done:
            # Take the action (index) that have the maximum expected future␣
↪reward given that state
            # we use epsilon=0 for exploitation

            action = agent.get_action(state,0)
            next_state, reward, terminated, truncated, info = env.step(action)

            total_rewards_ep += reward
            total_steps_ep+=1

            if reward == -10:
                total_penalties_ep += 1

            done=terminated or truncated

            state = next_state

        episode_rewards.append(total_rewards_ep)
        episode_steps.append(total_steps_ep)
        episode_penalties.append(total_penalties_ep)

    mean_reward = np.mean(episode_rewards)
    std_reward = np.std(episode_rewards)

    return mean_reward, std_reward, episode_rewards,episode_steps,␣
↪episode_penalties
```

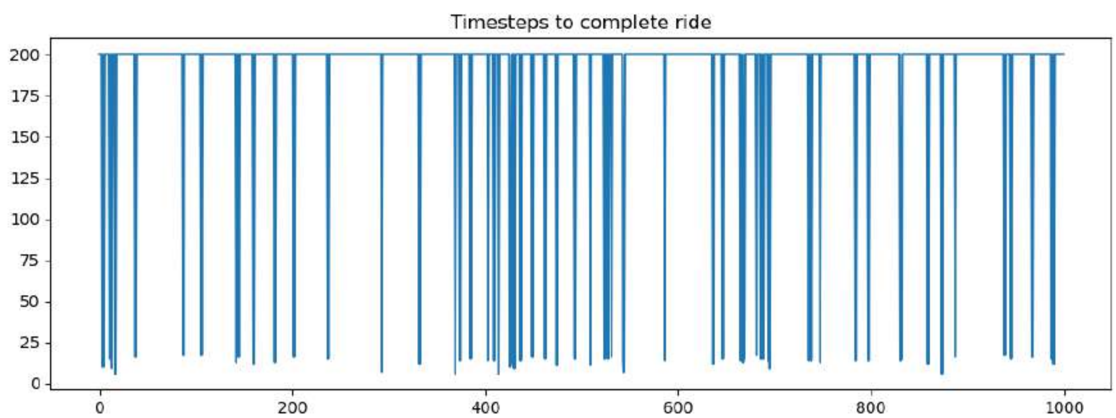
[13]:
```
mean_reward, std_reward,episode_rewards,episode_steps,␣
↪episode_penalties=evaluate_agent(env,agent, 1000)
print(f"Mean_reward={mean_reward:.2f} +/- {std_reward:.2f}")
```

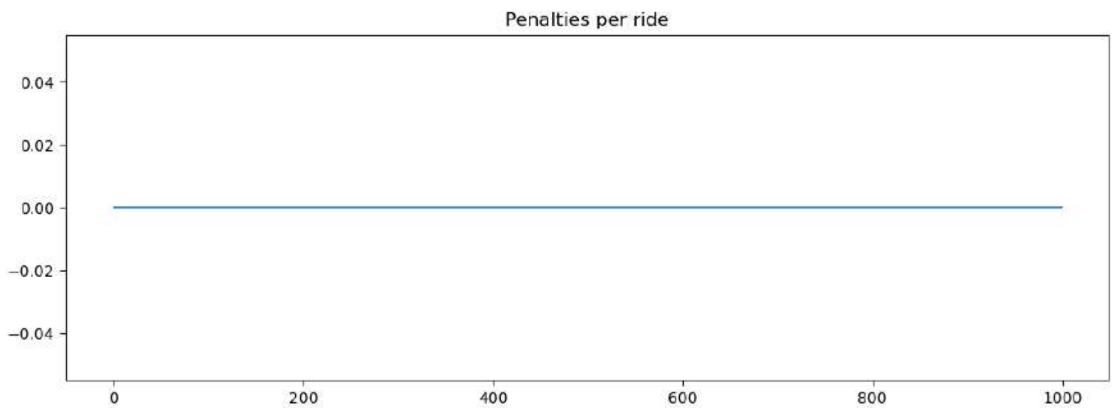
```
100%|      | 1000/1000 [00:03<00:00, 279.70it/s]

Mean_reward=-187.11 +/- 50.13
```

[14]:
```
fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Cumulative Rewards per ride")
pd.Series(episode_rewards).plot(kind='line')
```

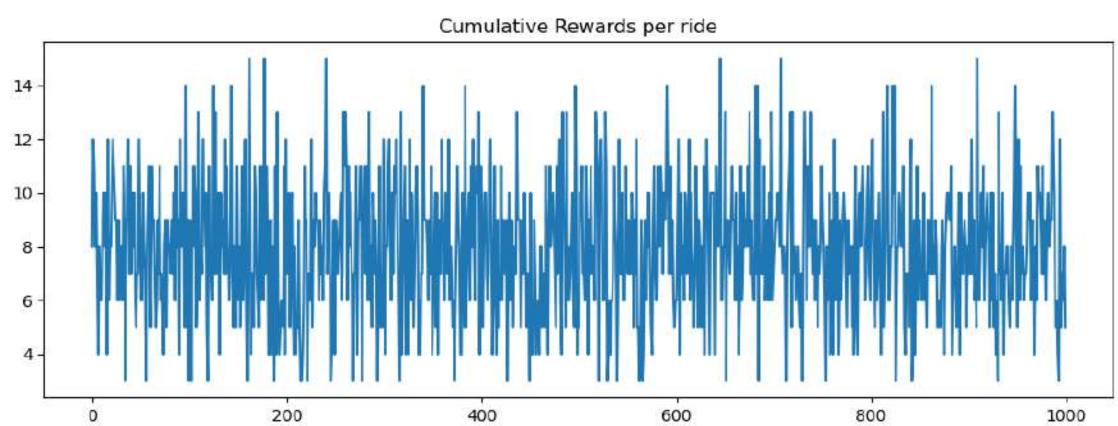
```
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Timesteps to complete ride")
pd.Series(episode_steps).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Penalties per ride")
pd.Series(episode_penalties).plot(kind='line')
plt.show()
```

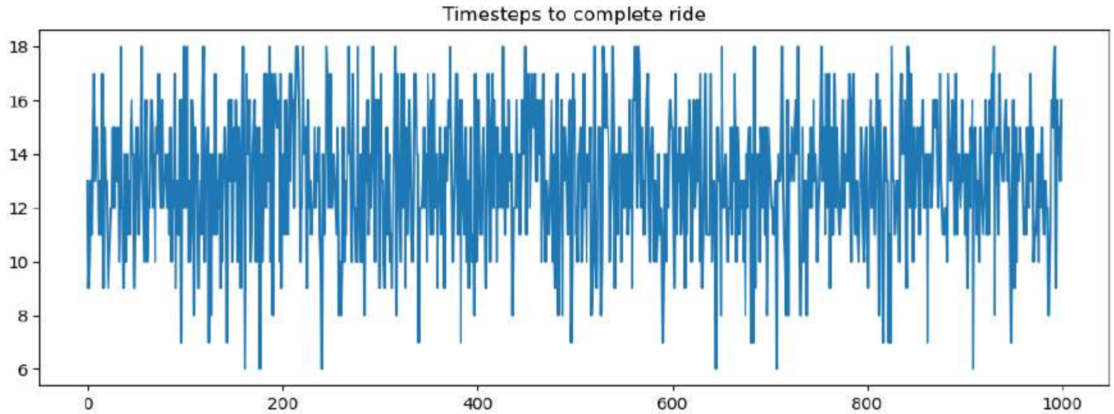


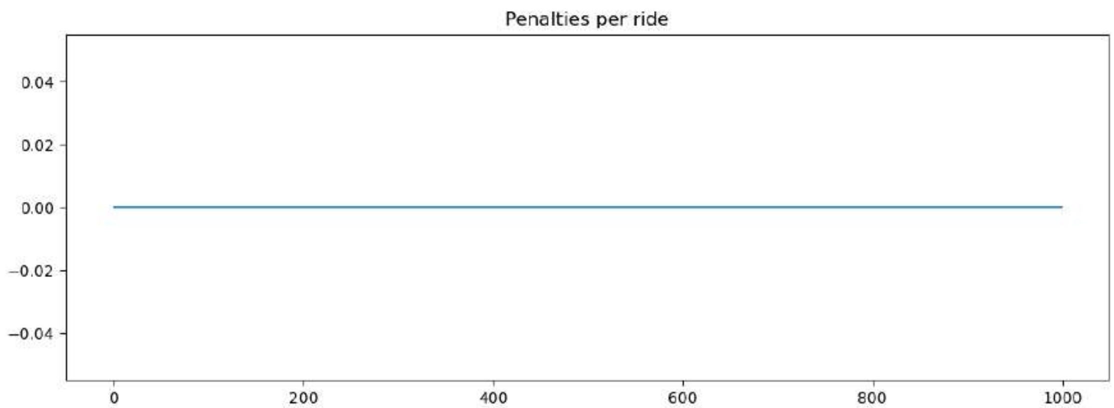




[117]:
```
fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Cumulative Rewards per ride")
pd.Series(episode_rewards).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Timesteps to complete ride")
pd.Series(episode_steps).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Penalties per ride")
pd.Series(episode_penalties).plot(kind='line')
plt.show()
```







3.0.2 Record a simulation as a video

[118]:
```
def record_video(env, agent, out_directory, fps=1):
    """
    Generate a replay video of the agent
    :param env
    :param agent:  agent within its Qtable
    :param out_directory
    :param fps: how many frame per seconds (with taxi-v3 and frozenlake-v1 we␣
 ↪use 1)
    """
    images = []
    terminated = False
    truncated = False
    state, info = env.reset(seed=random.randint(0,500))
    img = env.render()
    images.append(img)
```

```
    while not (terminated or truncated):
        # Take the action (index) that have the maximum expected future reward␣
↪given that state
        action = np.argmax(agent.q_table[state][:])
        state, reward, terminated, truncated, info = env.step(action) # We␣
↪directly put next_state = state for recording logic
        img = env.render()
        images.append(img)
    imageio.mimsave(out_directory, [np.array(img) for i, img in␣
↪enumerate(images)], fps=fps)
```

[120]:
```
from base64 import b64encode
from IPython.display import HTML

# generate the video
video_path = "./replay.mp4"
record_video(env, agent, video_path, 1)

# Show video
mp4 = open(video_path,'rb').read()
data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
HTML("""<video width=400 controls>      <source src="%s" type="video/mp4"></
↪video>""" % data_url)
```

```
IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by
macro_block_size=16, resizing from (550, 350) to (560, 352) to ensure video
compatibility with most codecs and players. To prevent resizing, make your input
image divisible by the macro_block_size or set the macro_block_size to 1
(risking incompatibility).
[swscaler @ 0x7f952e2d8000] Warning: data is not aligned! This can lead to a
speed loss
```

[120]: <IPython.core.display.HTML object>

[ ]:
```
# Show video
video_path = "./replay.mp4"
mp4 = open(video_path,'rb').read()
data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
HTML("""<video width=400 controls>      <source src="%s" type="video/mp4"></
↪video>""" % data_url)
```

wjbjj4om6

December 17, 2023

[2]:
```
!pip install -r https://raw.githubusercontent.com/malkiAbdelhamid/
↪Advanced-Deep-Learning-2023-2024-esisba/master/lab1_QLearning/
↪requirements_lab1.txt
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from -r https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 1))
(1.23.5)
Collecting gymnasium (from -r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 2))
  Downloading gymnasium-0.29.1-py3-none-any.whl (953 kB)
                          953.9/953.9 kB
17.2 MB/s eta 0:00:00
Requirement already satisfied: pygame in /usr/local/lib/python3.10/dist-
packages (from -r https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-
Deep-Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line
3)) (2.5.2)
Collecting jupyter (from -r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
  Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from -r https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 5))
(1.5.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from -r https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-
Deep-Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line
6)) (3.7.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from -r https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 7))
(4.66.1)
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-
packages (from -r https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-
Deep-Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line
8)) (2.31.5)
```

```
Requirement already satisfied: cloudpickle>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from gymnasium->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 2))
(2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in
/usr/local/lib/python3.10/dist-packages (from gymnasium->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 2))
(4.5.0)
Collecting farama-notifications>=0.0.1 (from gymnasium->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 2))
  Downloading Farama_Notifications-0.0.4-py3-none-any.whl (2.5 kB)
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-
packages (from jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(6.5.5)
Collecting qtconsole (from jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
  Downloading qtconsole-5.4.4-py3-none-any.whl (121 kB)
                          121.9/121.9 kB
13.3 MB/s eta 0:00:00
Requirement already satisfied: jupyter-console in
/usr/local/lib/python3.10/dist-packages (from jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(6.1.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-
packages (from jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(6.5.4)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-
packages (from jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(5.5.6)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-
packages (from jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(7.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
```

```
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 5))
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 5))
(2023.3.post1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 6))
(1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 6))
(0.12.0)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 6))
(4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 6))
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 6))
(23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 6))
(9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 6))
(3.1.1)
Requirement already satisfied: imageio-ffmpeg in /usr/local/lib/python3.10/dist-
packages (from imageio->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 8))
(0.4.9)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages
```

```
(from imageio->-r https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-
Deep-Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line
8)) (5.9.5)
Collecting av (from imageio->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 8))
  Downloading
av-10.0.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (31.0 MB)
                           31.0/31.0 MB
61.5 MB/s eta 0:00:00
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.1->pandas->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 5))
(1.16.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from imageio-ffmpeg->imageio->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 8))
(67.7.2)
Requirement already satisfied: ipython-genutils in
/usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.2.0)
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-
packages (from ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(7.34.0)
Requirement already satisfied: traitlets>=4.1.0 in
/usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(5.7.1)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-
packages (from ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(6.1.12)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-
packages (from ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(6.3.2)
Requirement already satisfied: widgetsnbextension~=3.6.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
```

```
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(3.6.6)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(3.0.9)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(3.0.39)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-
packages (from jupyter-console->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(2.16.1)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages
(from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(4.9.3)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-
packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(4.11.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages
(from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(6.0.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-
packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(3.1.2)
Requirement already satisfied: jupyter-core>=4.7 in
```

```
/usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(5.3.2)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.2.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(2.1.3)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.8.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-
packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(5.9.2)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.5.0)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-
packages (from nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.2.1)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-
packages (from notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(23.2.1)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-
packages (from notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
```

```
(23.1.0)
Requirement already satisfied: nest-asyncio>=1.5 in
/usr/local/lib/python3.10/dist-packages (from notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.5.8)
Requirement already satisfied: Send2Trash>=1.8.0 in
/usr/local/lib/python3.10/dist-packages (from notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.8.2)
Requirement already satisfied: terminado>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.17.1)
Requirement already satisfied: prometheus-client in
/usr/local/lib/python3.10/dist-packages (from notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.17.1)
Requirement already satisfied: nbclassic>=0.4.7 in
/usr/local/lib/python3.10/dist-packages (from notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.0.0)
Collecting qtpy>=2.4.0 (from qtconsole->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
  Downloading QtPy-2.4.0-py3-none-any.whl (93 kB)
                           93.4/93.4 kB
12.3 MB/s eta 0:00:00
Collecting jedi>=0.16 (from ipython>=5.0.0->ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
                           1.6/1.6 MB
98.1 MB/s eta 0:00:00
Requirement already satisfied: decorator in
/usr/local/lib/python3.10/dist-packages (from
ipython>=5.0.0->ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-
packages (from ipython>=5.0.0->ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
```

```
(0.7.5)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-
packages (from ipython>=5.0.0->ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.2.0)
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.10/dist-packages (from
ipython>=5.0.0->ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-
packages (from ipython>=5.0.0->ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(4.8.0)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-
core>=4.7->nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(3.11.0)
Requirement already satisfied: jupyter-server>=1.8 in
/usr/local/lib/python3.10/dist-packages (from
nbclassic>=0.4.7->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.24.0)
Requirement already satisfied: notebook-shim>=0.2.3 in
/usr/local/lib/python3.10/dist-packages (from
nbclassic>=0.4.7->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.2.3)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.10/dist-
packages (from nbformat>=5.1->nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(2.18.1)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from
nbformat>=5.1->nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(4.19.1)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-
packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->jupyter-
```

```
console->jupyter->-r https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-
Deep-Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line
4)) (0.2.8)
Requirement already satisfied: ptyprocess in /usr/local/lib/python3.10/dist-
packages (from terminado>=0.8.3->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.7.0)
Requirement already satisfied: argon2-cffi-bindings in
/usr/local/lib/python3.10/dist-packages (from argon2-cffi->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(21.2.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-
packages (from beautifulsoup4->nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(2.5)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-
packages (from bleach->nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.5.1)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from
jedi>=0.16->ipython>=5.0.0->ipykernel->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.8.3)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r
```

```
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(0.10.4)
Requirement already satisfied: anyio<4,>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(3.7.1)
Requirement already satisfied: websocket-client in
/usr/local/lib/python3.10/dist-packages (from jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.6.3)
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-
packages (from argon2-cffi-bindings->argon2-cffi->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.16.0)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-
packages (from anyio<4,>=3.1.0->jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(3.4)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-
packages (from anyio<4,>=3.1.0->jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.3.0)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-
packages (from anyio<4,>=3.1.0->jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(1.1.3)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-
packages (from cffi>=1.0.1->argon2-cffi-
bindings->argon2-cffi->notebook->jupyter->-r
https://raw.githubusercontent.com/malkiAbdelhamid/Advanced-Deep-
Learning-2023-2024-esisba/master/lab1_QLearning/requirements_lab1.txt (line 4))
(2.21)
Installing collected packages: farama-notifications, av, qtpy, jedi, gymnasium,
qtconsole, jupyter
Successfully installed av-10.0.0 farama-notifications-0.0.4 gymnasium-0.29.1
jedi-0.19.1 jupyter-1.0.0 qtconsole-5.4.4 qtpy-2.4.0
```

Now that we studied the Q-Learning algorithm, let's implement it from scratch and train our Q-Learning agent in Taxi-3 environment:

Q-Learning **is the RL algorithm that**:

- Trains *Q-Function*, an **action-value function** that encoded, in internal memory, by a *Q-table* **that contains all the state-action pair values.**
- Given a state and action, our Q-Function **will search the Q-table for the corresponding value.**
- When the training is done,**we have an optimal Q-Function, so an optimal Q-Table.**
- And if we **have an optimal Q-function**, we have an optimal policy, since we **know for, each state, the best action to take.**

0.0.1 Import the packages

[3]:
```
import gymnasium as gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from tqdm import tqdm
import imageio
```

0.0.2 QAgent Class:

Is initialized with the environment *env* and the necessary hyper-parameters (*alpha*, *gamma*, *epsilon*, etc)

This class implements the most important steps of **Q-Learning** Algorithm: - Q-Table initialization - Q-Table shape=(n_states * n_actions)

- Epsilon-greedy policy as an acting policy
  - With *probability 1-* : **we do exploitation** (i.e. our agent selects the action with the highest state-action pair value).
  - With *probability* : we do **exploration** (trying a random action).
- Greedy-policy as an updating policy
  - using Bellman equation *Q(s,a) + lr[R(s,a) + gamma * max Q(s',a') - Q(s,a)]*

[4]:
```
class QAgent:

    def __init__(self, env, alpha, gamma, max_epsilon, min_epsilon, decay_rate):

        self.env = env

        # table with q-values: n_states * n_actions
        self.q_table = np.zeros(shape=(env.observation_space.n, env.
↪action_space.n))
```

```python
        # hyper-parameters
        self.alpha = alpha                          # learning rate
        self.gamma = gamma                          # discount factor
        self.max_epsilon = max_epsilon              # Exploration probability at␣
↪start
        self.min_epsilon = min_epsilon              # Minimum exploration␣
↪probability
        self.decay_rate =  decay_rate               # Exponential decay rate for␣
↪exploration prob



    # get action using epsilon greedy policy
    def get_action(self, state, epsilon):
        # Randomly generate a number between 0 and 1
        random_num =random.random()

        # if random_num > greater than epsilon --> exploitation
        if random_num > epsilon:
            # Take the action with the highest value given a state
            action = np.argmax(self.q_table[state, :])

        # else --> exploration
        else:
             action = self.env.action_space.sample()

        return action

    def update_parameters(self, state, action, reward, next_state):
        """"""
        # Q-learning formula
        # Update Q(s,a):= Q(s,a) + lr [R(s,a)  + gamma * max Q(s',a') - Q(s,a)]
        old_value = self.q_table[state, action]
        next_max = np.argmax(self.q_table[next_state, :])
        new_value = old_value + self.alpha * (reward + self.gamma*next_max -␣
↪old_value)

        # update the q_table
        self.q_table[state, action] = new_value
```

0.0.3 The training loop:

The training is based on *Temporal Difference (TD) learning* where the Q-Table is updated after each step

- For episode in the total of training episodes:
    - Reduce epsilon (since we need less and less exploration)

- Reset the environment
- For step in max timesteps:
  * Choose the action a using epsilon greedy policy
  * Take the action (a) and observe the outcome state(s') and reward (r)
  * Update the Q-value Q(s,a) using Bellman equation Q(s,a) + lr [R(s,a) + gamma * max Q(s',a') - Q(s,a)]
  * If done, finish the episode
  * Our next state is the new state

[5]:
```
def train(n_training_episodes,  env, agent):

    episode_rewards = []
    episode_penalties = []
    episode_steps = []
    for episode in tqdm(range(n_training_episodes)):

        total_rewards_ep = 0
        total_penalties_ep=0
        total_steps_ep=0

        # Reduce epsilon (because we need less and less exploration)
        epsilon = agent.min_epsilon + (agent.max_epsilon - agent.
 ↪min_epsilon)*np.exp(-agent.decay_rate*episode)

        # Reset the environment
        state, info = env.reset()
        step = 0
        terminated = False
        truncated = False
        done=False

        # repeat
        while not done:
            # Choose the action At using epsilon greedy policy
            action = agent.get_action(state, epsilon)

            # Take action At and observe Rt+1 and St+1
            # Take the action (a) and observe the outcome state(s') and reward␣
 ↪(r)
            next_state, reward, terminated, truncated, info = env.step(action)

            total_rewards_ep += reward
            total_steps_ep +=1
            if reward == -10:
                total_penalties_ep += 1
```

```
            # Update Q(s,a):= Q(s,a) + lr [R(s,a)  + gamma * max Q(s',a') -
↪Q(s,a)]
            agent.update_parameters(state, action, reward, next_state)

            # If terminated or truncated finish the episode
            done=terminated or truncated

            if done:
                break

            # Our next state is the new state
            #--- add code here----------
            state = next_state

        episode_rewards.append(total_rewards_ep)
        episode_steps.append(total_steps_ep)
        episode_penalties.append(total_penalties_ep)

    return  episode_rewards,episode_steps, episode_penalties
```

0.0.4 Hyper-Parameters

The exploration related hyper-paramters are some of the most important ones. - We need to make sure that our agent **explores enough of the state space** to learn a good value approximation. To do that, we need to have progressive decay of the epsilon. - If you decrease epsilon too fast (too high decay_rate), **you take the risk that your agent will be stuck**, since your agent didn't explore enough of the state space and hence can't solve the problem.

[6]:
```
# Training parameters
n_training_episodes = 1000  # Total training episodes
alpha= 0.7                   # Learning rate

# Environment parameters
gamma = 0.95                 # Discounting rate

# Exploration parameters
max_epsilon = 1.0             # Exploration probability at start
min_epsilon = 0.05            # Minimum exploration probability
decay_rate = 0.0005           # Exponential decay rate for exploration prob

# Evaluation parameters
n_eval_episodes = 100        # Total number of test episodes
```

0.0.5 Train the Q-Agent on Taxiv3 Environment

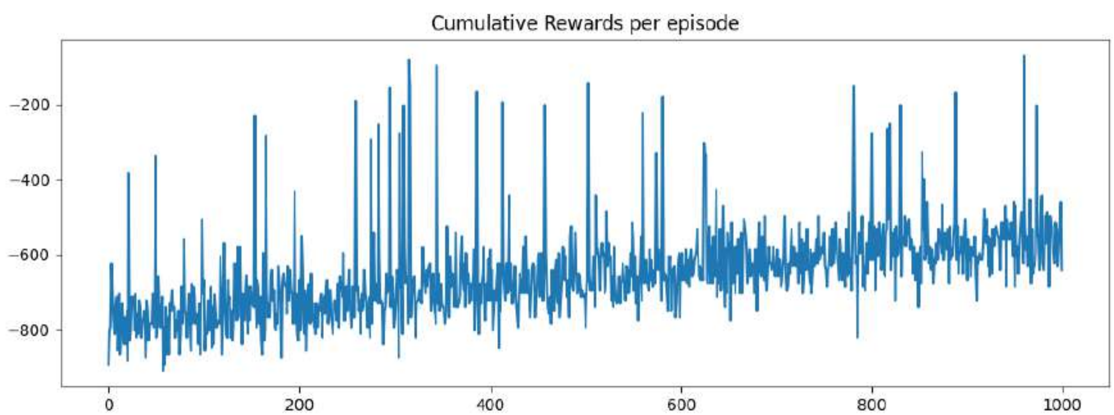
[7]:
```
env=gym.make("Taxi-v3", render_mode="rgb_array")

agent=QAgent(env, alpha, gamma, max_epsilon, min_epsilon, decay_rate)

episode_rewards,episode_steps, episode_penalties=train(n_training_episodes,␣
↪env,agent)
```

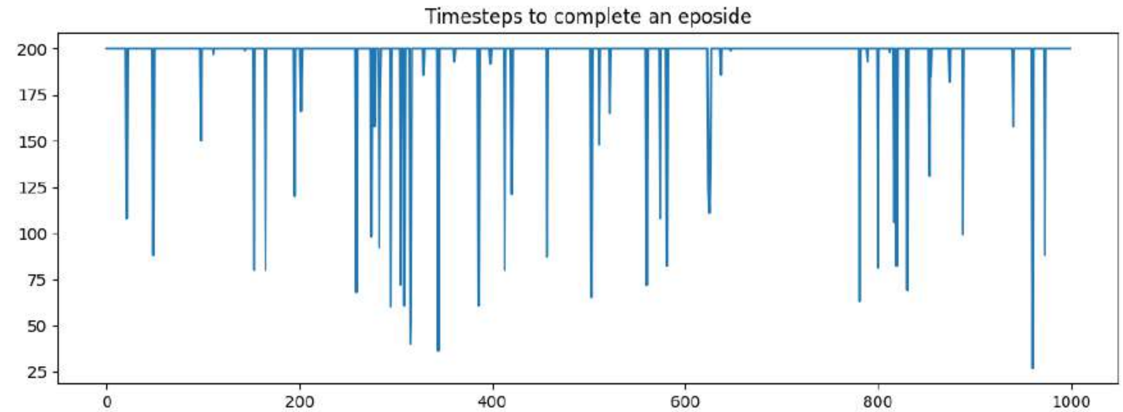
```
100%|      | 1000/1000 [00:05<00:00, 177.34it/s]
```

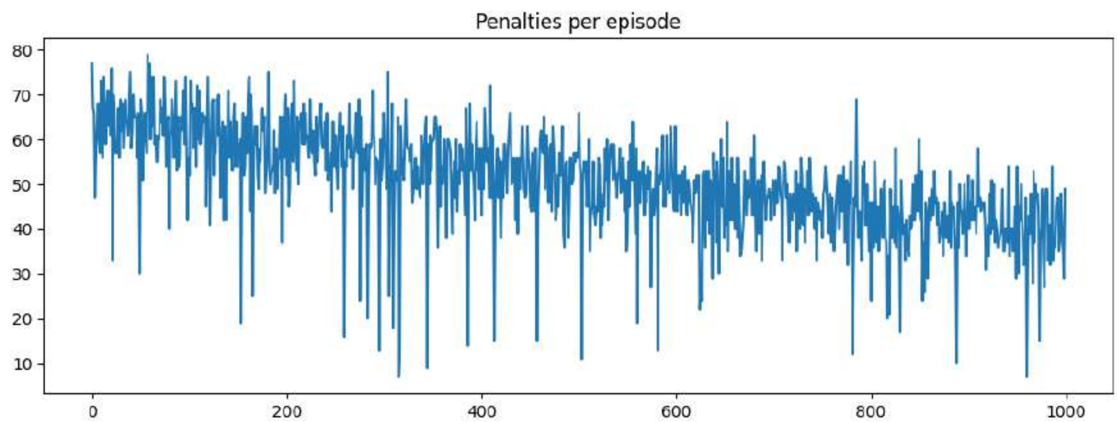
0.0.6 Plot the training result

- episode_rewards: the cumulative rewards progression over all episodes (should be increased over time)
- episode_steps: the required step per episode (should be decreased over time)
- episode_penalties: the total penalties per episode (should be decreased over time)

[8]:
```
fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Cumulative Rewards per episode")
pd.Series(episode_rewards).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Timesteps to complete an eposide")
pd.Series(episode_steps).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Penalties per episode")
pd.Series(episode_penalties).plot(kind='line')
plt.show()
```







0.0.7 Evaluate the QAgent after training

[9]:
```python
import pickle
file = open('trained_taxi', 'wb')

# dump information to that file
pickle.dump(agent, file)

# close the file
file.close()
```

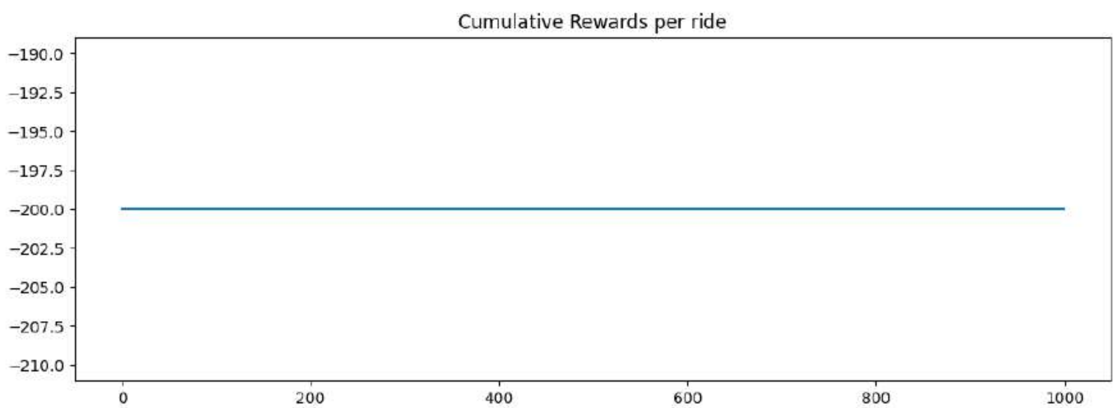
[10]:
```python
def evaluate_agent(env,agent,  n_eval_episodes):
    """
    Evaluate the agent for ``n_eval_episodes`` episodes and returns average␣
 ↪reward and std of reward.
    :param agent: the gent within its  evaluation environment and Qtable
    :param max_steps: Maximum number of steps per episode
```

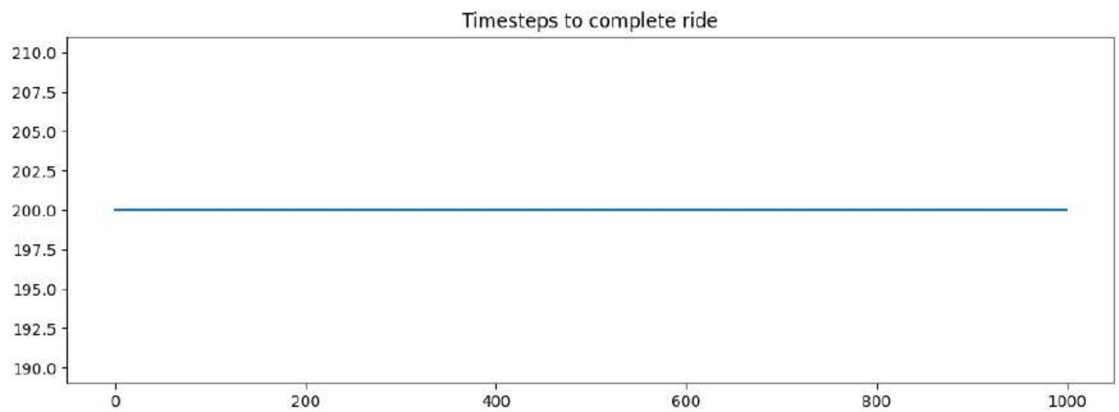
```
    :param n_eval_episodes: Number of episode to evaluate the agent
    """
    episode_rewards = []
    episode_penalties = []
    episode_steps = []
    for episode in tqdm(range(n_eval_episodes)):

      state, info= env.reset()
      step = 0
      truncated = False
      terminated = False
      total_rewards_ep = 0
      total_penalties_ep=0
      total_steps_ep=0
      done= False

      while not done:
        # Take the action (index) that have the maximum expected future reward␣
↪given that state
        # we use epsilon=0 for exploitation

        action = agent.get_action(state,0)
        next_state, reward, terminated, truncated, info = env.step(action)

        total_rewards_ep += reward
        total_steps_ep+=1

        if reward == -10:
              total_penalties_ep += 1

        done=terminated or truncated

        if done:
          break

        state = next_state

      episode_rewards.append(total_rewards_ep)
      episode_steps.append(total_steps_ep)
      episode_penalties.append(total_penalties_ep)

    mean_reward = np.mean(episode_rewards)
    std_reward = np.std(episode_rewards)

    return mean_reward, std_reward, episode_rewards,episode_steps,␣
↪episode_penalties
```

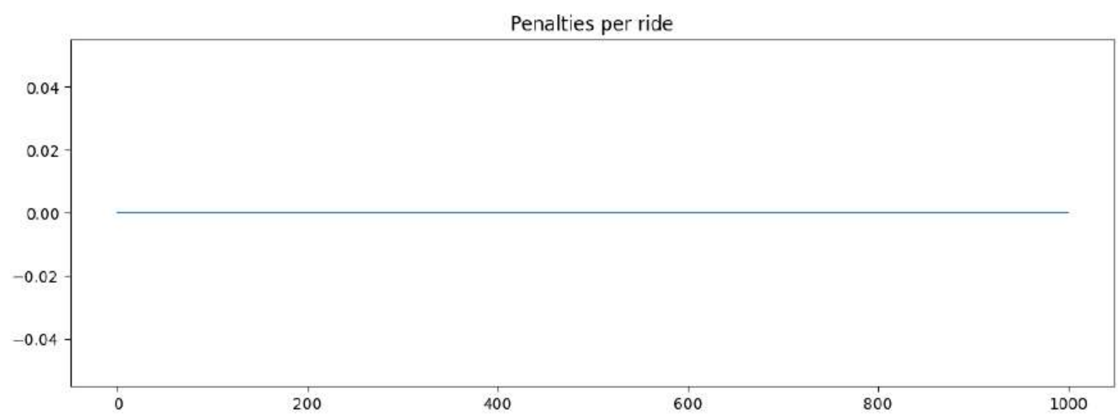
[11]:
```
mean_reward, std_reward,episode_rewards,episode_steps,␣
↪episode_penalties=evaluate_agent(env,agent, 1000)
print(f"Mean_reward={mean_reward:.2f} +/- {std_reward:.2f}")
```

```
100%|      | 1000/1000 [00:04<00:00, 237.69it/s]

Mean_reward=-200.00 +/- 0.00
```

[12]:
```
fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Cumulative Rewards per ride")
pd.Series(episode_rewards).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Timesteps to complete ride")
pd.Series(episode_steps).plot(kind='line')
plt.show()

fig, ax = plt.subplots(figsize = (12, 4))
ax.set_title("Penalties per ride")
pd.Series(episode_penalties).plot(kind='line')
plt.show()
```







0.0.8 Record a simulation as a video

[13]:
```
def record_video(env, agent, out_directory, fps=1):
    """
    Generate a replay video of the agent
    :param env
    :param agent:  agent within its Qtable
    :param out_directory
    :param fps: how many frame per seconds (with taxi-v3 and frozenlake-v1 we use␣
↪1)
    """
    images = []
    terminated = False
    truncated = False
    state, info = env.reset(seed=random.randint(0,500))
    img = env.render()
    images.append(img)
```

```
    while not (terminated or truncated):
      # Take the action (index) that have the maximum expected future reward␣
↪given that state
      action = np.argmax(agent.q_table[state][:])
      state, reward, terminated, truncated, info = env.step(action) # We directly␣
↪put next_state = state for recording logic
      img = env.render()
      images.append(img)
    imageio.mimsave(out_directory, [np.array(img) for i, img in␣
↪enumerate(images)], fps=fps)
```

[14]:
```
from base64 import b64encode
from IPython.display import HTML

# generate the video
video_path = "./replay.mp4"
record_video(env, agent, video_path, 1)

# Show video
mp4 = open(video_path,'rb').read()
data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
HTML("""<video width=400 controls>      <source src="%s" type="video/mp4"></
↪video>""" % data_url)
```

```
WARNING:imageio_ffmpeg:IMAGEIO FFMPEG_WRITER WARNING: input image is not
divisible by macro_block_size=16, resizing from (550, 350) to (560, 352) to
ensure video compatibility with most codecs and players. To prevent resizing,
make your input image divisible by the macro_block_size or set the
macro_block_size to 1 (risking incompatibility).
```

[14]: <IPython.core.display.HTML object>

[ ]:
```
# Show video
mp4 = open(video_path,'rb').read()
data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
HTML("""<video width=400 controls>      <source src="%s" type="video/mp4"></
↪video>""" % data_url)
```

xj4nrszcf

December 17, 2023

0.1 ## Deep Q-Network (DQN) Vs Deep Q-Network without Target

In this notebook, you will implement two versions of Deep Q-Learning agent: - DQN with a fixed Target network - DQN without Target network

The two agents will be trained with OpenAI Gym's CartPole_v1 environment.

0.1.1 Import the Necessary Packages

[1]:
```
import gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from tqdm import tqdm
import imageio
from collections import deque, namedtuple

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

0.1.2 Define some hyperparameter

[2]:
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[3]:
```python
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

0.1.3 Define Neural Network Architecture.

Since `CartPole_v1` environment is sort of simple envs, we don't need complicated architecture. We just need non-linear function approximator that maps from state to action.

[4]:
```python
class QNetwork(nn.Module):

    def __init__(self, state_shape, action_space_size, seed):
        """Initialize parameters and build model.
        Params
        ======
            state_shape (int): Dimension of each state
            action_space_size (int): Dimension of each action
            seed (int): Random seed
        """
        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_shape, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, action_space_size)

    def forward(self, state):
        """Build a network that maps state -> action values."""
        x = self.fc1(state)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x=self.fc3(x)
        return x
```

0.1.4 Define Replay Buffer

0.1.5 Experience Replay

To perform *experience replay* the authors store the agent's experiences e_t as represented by the tuple

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

consisting of the observed state in period t, the reward received in period t, the action taken in period t, and the resulting state in period $t+1$. The dataset of agent experiences at period t consists of the set of past experiences.

$$D_t = \{e1, e2, ..., e_t\}$$

Depending on the task it may note be feasible for the agent to store the entire history of past experiences.

During learning Q-learning updates are computed based on samples (or minibatches) of experience (s, a, r, s'), drawn uniformly at random from the pool of stored samples D_t.

The following is my Python implmentation of these ideas.

[5]:
```
class ReplayBuffer:
    """Fixed-size buffer to store experience tuples."""

    def __init__(self, buffer_size, batch_size, seed):
        """Initialize a ReplayBuffer object.

        Params
        ======
            buffer_size (int): maximum size of buffer
            batch_size (int): size of each training batch
            seed (int): random seed
        """
        self.memory = deque(maxlen=buffer_size)
        self.batch_size = batch_size
        self.experience = namedtuple("Experience", field_names=["state",␣
↪"action", "reward", "next_state", "done"])
        self.seed = random.seed(seed)

    def add(self, state, action, reward, next_state, done):
        """Add a new experience to memory."""
        e = self.experience(state, action, reward, next_state, done)
        self.memory.append(e)

    def sample(self):
        """Randomly sample a batch of experiences from memory."""
        experiences = random.sample(self.memory, k=self.batch_size)

        states = torch.from_numpy(np.vstack([e.state for e in experiences if e␣
↪is not None])).float().to(device)
        actions = torch.from_numpy(np.vstack([e.action for e in experiences if␣
↪e is not None])).long().to(device)
        rewards = torch.from_numpy(np.vstack([e.reward for e in experiences if␣
↪e is not None])).float().to(device)
        next_states = torch.from_numpy(np.vstack([e.next_state for e in␣
↪experiences if e is not None])).float().to(device)
        dones = torch.from_numpy(np.vstack([e.done for e in experiences if e is␣
↪not None]).astype(np.uint8)).float().to(device)

        return (states, actions, rewards, next_states, dones)
```

```python
    def __len__(self):
        """Return the current size of internal memory."""
        return len(self.memory)
```

0.1.6 Define the Deep QLearning Agent

The Deep Q-learning update at iteration i uses the following loss function

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$

where γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the Q-network parameters used to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters θ_i every C steps and are held fixed between individual updates.

[6]:
```python
class DQAgent():
    """Interacts with and learns from the environment."""

    def __init__(self, state_shape, action_space_size, seed):
        """Initialize an Agent object.

        Params
        ======
            state_shape (int): dimension of each state
            action_space_size (int): dimension of each action
            seed (int): random seed
        """
        self.state_shape = state_shape
        self.action_space_size = action_space_size
        self.seed = random.seed(seed)

        # Q-Network
        self.qnetwork_local = QNetwork(state_shape, action_space_size, seed).
↪to(device)
        self.qnetwork_target = QNetwork(state_shape, action_space_size, seed).
↪to(device)

        self.optimizer = optim.Adam(self.qnetwork_local.parameters(), lr=LR)

        # Replay memory
        self.memory = ReplayBuffer(BUFFER_SIZE, BATCH_SIZE, seed)

        # Initialize time step (for updating every UPDATE_EVERY steps)
        self.t_step = 0

    def step(self, state, action, reward, next_state, done):
```

```
        # Save experience in replay memory
        self.memory.add(state, action, reward, next_state, done)

        # Learn every UPDATE_EVERY time steps.
        self.t_step = (self.t_step + 1) % UPDATE_EVERY
        if self.t_step == 0:
            # If enough samples are available in memory, get random subset and learn
            if len(self.memory) > BATCH_SIZE:
                experiences = self.memory.sample()
                self.learn(experiences, GAMMA)

    def act(self, state, eps=0.):
        """Returns actions for given state as per current policy.
        Params
        ======
            state (array_like): current state
            eps (float): epsilon, for epsilon-greedy action selection
        """
        # Epsilon-greedy action selection
        if random.random() > eps:
            state = torch.from_numpy(state).float().unsqueeze(0).to(device)
            action_values = self.qnetwork_local(state)
            return np.argmax(action_values.cpu().data.numpy())
        else:
            return random.choice(np.arange(self.action_space_size))

    def learn(self, experiences, gamma):
        """Update value parameters using given batch of experience tuples.

        Params
        ======
            experiences (Tuple[torch.Variable]): tuple of (s, a, r, s', done) tuples
            gamma (float): discount factor
        """
        # Obtain random minibatch of tuples from D
        states, actions, rewards, next_states, dones = experiences

        ## Compute and minimize the loss
        ### Extract next maximum estimated value from target network
        q_targets_next = self.qnetwork_target(next_states).detach().max(1)[0].unsqueeze(1)

        ### Calculate target value from bellman equation
        q_targets = rewards + gamma * q_targets_next * (1 - dones)
```

```
        ### Calculate expected value from local network
        q_expected = self.qnetwork_local(states).gather(1, actions)

        ### Loss calculation (we used Mean squared error)
        loss = F.mse_loss(q_expected, q_targets)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # ------------------- update target network ------------------- #
        self.soft_update(self.qnetwork_local, self.qnetwork_target, TAU)

    def soft_update(self, local_model, target_model, tau):
        """Soft update model parameters.
        _target =  *_local + (1 -  )*_target

        Params
        ======
            local_model (PyTorch model): weights will be copied from
            target_model (PyTorch model): weights will be copied to
            tau (float): interpolation parameter
        """
        for target_param, local_param in zip(target_model.parameters(),␣
↪local_model.parameters()):
            target_param.data.copy_(tau*local_param.data + (1.
↪0-tau)*target_param.data)
```

0.1.7 Define the Deep QLearning Agent Without Target Network

[7]:
```
class DQAgent_Without_Target():
    """Interacts with and learns from the environment."""

    def __init__(self, state_shape, action_space_size, seed):
        """Initialize an Agent object.

        Params
        ======
            state_shape (int): dimension of each state
            action_space_size (int): dimension of each action
            seed (int): random seed
        """
        self.state_shape = state_shape
        self.action_space_size = action_space_size
        self.seed = random.seed(seed)

        # Q-Network
```

```python
        self.qnetwork_local = QNetwork(state_shape, action_space_size, seed).
↪to(device)
        self.qnetwork_target = QNetwork(state_shape, action_space_size, seed).
↪to(device)

        self.optimizer = optim.Adam(self.qnetwork_local.parameters(), lr=LR)

        # Replay memory
        self.memory = ReplayBuffer(BUFFER_SIZE, BATCH_SIZE, seed)

        # Initialize time step (for updating every UPDATE_EVERY steps)
        self.t_step = 0

    def step(self, state, action, reward, next_state, done):
        # Save experience in replay memory
        self.memory.add(state, action, reward, next_state, done)

        # Learn every UPDATE_EVERY time steps.
        self.t_step = (self.t_step + 1) % UPDATE_EVERY
        if self.t_step == 0:
            # If enough samples are available in memory, get random subset and␣
↪learn
            if len(self.memory) > BATCH_SIZE:
                experiences = self.memory.sample()
                self.learn(experiences, GAMMA)

    def act(self, state, eps=0.):
        """Returns actions for given state as per current policy.
        Params
        ======
            state (array_like): current state
            eps (float): epsilon, for epsilon-greedy action selection
        """
        # Epsilon-greedy action selection
        if random.random() > eps:
            state = torch.from_numpy(state).float().unsqueeze(0).to(device)
            action_values = self.qnetwork_local(state)
            return np.argmax(action_values.cpu().data.numpy())
        else:
            return random.choice(np.arange(self.action_space_size))

    def learn(self, experiences, gamma):
        """Update value parameters using given batch of experience tuples.

        Params
        ======
```

```
            experiences (Tuple[torch.Variable]): tuple of (s, a, r, s', done)␣
↪tuples
            gamma (float): discount factor
        """
        # Obtain random minibatch of tuples from D
        states, actions, rewards, next_states, dones = experiences

        ## Compute and minimize the loss
        ### Extract next maximum estimated value from target network
        q_local_next = self.qnetwork_local(next_states).detach().max(1)[0].
↪unsqueeze(1)

        ### Calculate target value from bellman equation
        q_targets = rewards + gamma * q_local_next * (1 - dones)

        ### Calculate expected value from local network
        q_expected = self.qnetwork_local(states).gather(1, actions)

        ### Loss calculation (we used Mean squared error)
        loss = F.mse_loss(q_expected, q_targets)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
```

0.1.8 Training Process

The code for the training loop remains unchanged For both agents (DQN and DQN Without Target).

[8]:
```
def dqn(agent, n_episodes=2500, max_t=1000, eps_start=1.0, eps_end=0.01,␣
↪eps_decay=0.995):
    """Deep Q-Learning.

    Params
    ======
        n_episodes (int): maximum number of training episodes
        max_t (int): maximum number of timesteps per episode
        eps_start (float): starting value of epsilon, for epsilon-greedy action␣
↪selection
        eps_end (float): minimum value of epsilon
        eps_decay (float): multiplicative factor (per episode) for decreasing␣
↪epsilon
    """
    scores = []                        # list containing scores from each␣
↪episode
    scores_window = deque(maxlen=100)  # last 100 scores
    eps = eps_start                    # initialize epsilon
```

```
    for i_episode in range(1, n_episodes+1):
        state = env.reset()
        score = 0
        for t in range(max_t):
            action = agent.act(state, eps)
            next_state, reward, done, _ = env.step(action)
            agent.step(state, action, reward, next_state, done)
            state = next_state
            score += reward
            if done:
                break
        scores_window.append(score)       # save most recent score
        scores.append(score)              # save most recent score
        eps = max(eps_end, eps_decay*eps) # decrease epsilon

        print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.
↪mean(scores_window)), end="")
        if i_episode % 100 == 0:
            print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.
↪mean(scores_window)))

    torch.save(agent.qnetwork_local.state_dict(), 'checkpoint.pth')
    return scores
```

0.1.9 Creating the Gym environment CartPole-v1

This environment is part of the Classic Control environments which contains general information about the environment.

0.1.10 Description

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson in "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem". A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.

0.1.11 Action Space

The action is a ndarray with shape (1,) which can take values {0, 1} indicating the direction of the fixed force the cart is pushed with.

- 0: Push cart to the left
- 1: Push cart to the right

Note: The velocity that is reduced or increased by the applied force is not fixed and it depends on the angle the pole is pointing. The center of gravity of the pole varies the amount of energy needed to move the cart underneath it

0.1.12 Observation Space

The observation is a ndarray with shape (4,) with the values corresponding to the following positions and velocities:

- Cart Position
- Cart Velocity
- Pole Angle
- Pole Angular Velocity

0.1.13 Rewards

Since the goal is to keep the pole upright for as long as possible, a reward of +1 for every step taken, including the termination step, is allotted. The threshold for rewards is 500

[9]:
```
env = gym.make('CartPole-v1')
print(env.reset())
print('State shape: ', env.observation_space.shape[0])
print('Number of actions: ', env.action_space.n)
```

```
[ 0.02176407 -0.02655943  0.03542723  0.01530834]
State shape:  4
Number of actions:  2
```

```
/usr/local/lib/python3.10/dist-packages/gym/core.py:317: DeprecationWarning:
WARN: Initializing wrapper in old step API which returns one bool instead

of two. It is recommended to set `new_step_api=True` to use new step API. This

will be the default behaviour in future.
  deprecation(
/usr/local/lib/python3.10/dist-
packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning:
WARN: Initializing environment in old step API which returns one bool

instead of two. It is recommended to set `new_step_api=True` to use new step

API. This will be the default behaviour in future.
  deprecation(
```

0.1.14 Training the DQagent using DQN

[10]:
```
load = False
```

[12]:
```
DQagent = DQAgent(state_shape=env.observation_space.shape[0],␣
↪action_space_size=env.action_space.n, seed=0)
if not load:
    scores_dqn = dqn(DQagent)
```

```
    torch.save(DQagent.qnetwork_local.state_dict(), "/content/
↪qnetwork_local-DQAgent.pth")
    torch.save(DQagent.qnetwork_target.state_dict(), "/content/
↪qnetwork_target-DQAgent.pth")
else:
    DQagent.qnetwork_local.load_state_dict(torch.load("/content/
↪qnetwork_local-DQAgent.pth"))
    DQagent.qnetwork_target.load_state_dict(torch.load("/content/
↪qnetwork_target-DQAgent.pth"))
```

0.1.15 Training the DQagent_Without_Target by avoiding the Target network

[13]:
```
DQagent_Without_Target = DQAgent_Without_Target(state_shape=env.
↪observation_space.shape[0], action_space_size=env.action_space.n, seed=0)
if not load:
    scores_dqn_without_target = dqn(DQagent_Without_Target)
    torch.save(DQagent_Without_Target.qnetwork_local.state_dict(), "/content/
↪qnetwork_local-DQagent_Without_Target.pth")
else:
    DQagent_Without_Target.qnetwork_local.load_state_dict(torch.load("/content/
↪qnetwork_local-DQagent_Without_Target.pth"))
```

```
Episode 100     Average Score: 17.52
Episode 200     Average Score: 43.40
Episode 300     Average Score: 244.61
Episode 400     Average Score: 307.86
Episode 500     Average Score: 330.34
Episode 600     Average Score: 305.01
Episode 700     Average Score: 85.94
Episode 800     Average Score: 9.86
Episode 900     Average Score: 21.21
Episode 1000    Average Score: 481.64
Episode 1100    Average Score: 495.50
Episode 1200    Average Score: 258.35
Episode 1300    Average Score: 366.34
Episode 1400    Average Score: 437.75
Episode 1500    Average Score: 56.54
Episode 1600    Average Score: 11.12
Episode 1700    Average Score: 9.46
Episode 1800    Average Score: 15.42
Episode 1900    Average Score: 213.74
Episode 2000    Average Score: 470.87
Episode 2100    Average Score: 223.82
Episode 2200    Average Score: 285.98
Episode 2300    Average Score: 493.18
Episode 2400    Average Score: 338.85
Episode 2500    Average Score: 11.41
```

0.2 Comparing DQN and DQN-Without_Target

Plotting the time series of scores (scores_dqn & scores_dqn_without_target) I can use Pandas to quickly plot the time series of scores along with a 100 episode moving average. Note that training stops as soon as the rolling average crosses the target score.

[14]:
```
scores_dqn   = pd.Series(scores_dqn , name="scores")
scores_dqn.describe()
```

[14]:
```
count    2500.000000
mean      134.790400
std       110.387981
min         8.000000
25%        12.000000
50%       179.000000
75%       212.000000
max       500.000000
Name: scores, dtype: float64
```

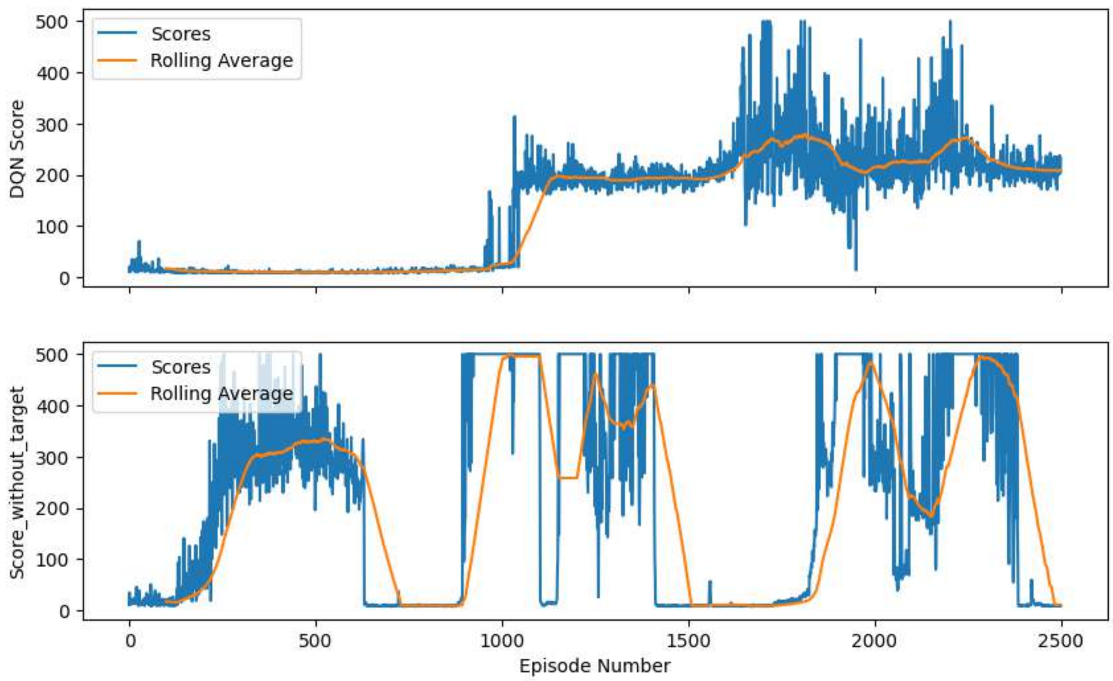
[15]:
```
scores_dqn_without_target   = pd.Series(scores_dqn_without_target ,␣
  ↪name="scores")
scores_dqn_without_target.describe()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[15]:
```
count    2500.000000
mean      221.428800
std       201.691669
min         8.000000
25%        11.000000
50%       222.000000
75%       468.000000
max       500.000000
Name: scores, dtype: float64
```

[17]:
```
fig, ax = plt.subplots(2, 1, figsize=(10, 6), sharex=True, sharey=True)
_ = scores_dqn.plot(ax=ax[0], label="Scores")
_ = (scores_dqn.rolling(window=100)
           .mean()
           .rename("Rolling Average")
           .plot(ax=ax[0]))
_=ax[0].legend()
_=ax[0].set_ylabel("DQN Score")
```

```
_ = scores_dqn_without_target.plot(ax=ax[1], label="Scores")
_ = (scores_dqn_without_target.rolling(window=100)
           .mean()
           .rename("Rolling Average")
           .plot(ax=ax[1]))
_=ax[1].legend()
_ = ax[1].set_ylabel("Score_without_target")
_ = ax[1].set_xlabel("Episode Number")
```



xe1i1tad6

December 17, 2023

0.1 ## Deep Q-Network (DQN) Vs Deep Q-Network without Experience Replay

In this notebook, you will implement two versions of Deep Q-Learning agent: - DQN with Experience Replay - DQN without Experience Replay

The two agents will be trained with OpenAI Gym's CartPole_v1 environment.

0.1.1 Import the Necessary Packages

[1]:
```
import gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from tqdm import tqdm
import imageio
from collections import deque, namedtuple

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

0.1.2 Define some hyperparameter

[2]:
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[3]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

0.1.3 Define Neural Network Architecture.

Since `CartPole_v1` environment is sort of simple envs, we don't need complicated architecture. We just need non-linear function approximator that maps from state to action.

[4]:
```
class QNetwork(nn.Module):

    def __init__(self, state_shape, action_space_size, seed):
        """Initialize parameters and build model.
        Params
        ======
            state_shape (int): Dimension of each state
            action_space_size (int): Dimension of each action
            seed (int): Random seed
        """
        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_shape, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, action_space_size)

    def forward(self, state):
        """Build a network that maps state -> action values."""
        x = self.fc1(state)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x=self.fc3(x)
        return x
```

0.1.4 Define Replay Buffer

0.1.5 Experience Replay

To perform *experience replay* the authors store the agent's experiences e_t as represented by the tuple

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

consisting of the observed state in period t, the reward received in period t, the action taken in period t, and the resulting state in period $t+1$. The dataset of agent experiences at period t consists of the set of past experiences.

$$D_t = \{e1, e2, ..., e_t\}$$

Depending on the task it may note be feasible for the agent to store the entire history of past experiences.

During learning Q-learning updates are computed based on samples (or minibatches) of experience (s, a, r, s'), drawn uniformly at random from the pool of stored samples D_t.

The following is my Python implmentation of these ideas.

[5]:
```
class ReplayBuffer:
    """Fixed-size buffer to store experience tuples."""

    def __init__(self, buffer_size, batch_size, seed):
        """Initialize a ReplayBuffer object.

        Params
        ======
            buffer_size (int): maximum size of buffer
            batch_size (int): size of each training batch
            seed (int): random seed
        """
        self.memory = deque(maxlen=buffer_size)
        self.batch_size = batch_size
        self.experience = namedtuple("Experience", field_names=["state",␣
↪"action", "reward", "next_state", "done"])
        self.seed = random.seed(seed)

    def add(self, state, action, reward, next_state, done):
        """Add a new experience to memory."""
        e = self.experience(state, action, reward, next_state, done)
        self.memory.append(e)

    def sample(self):
        """Randomly sample a batch of experiences from memory."""
        experiences = random.sample(self.memory, k=self.batch_size)

        states = torch.from_numpy(np.vstack([e.state for e in experiences if e␣
↪is not None])).float().to(device)
        actions = torch.from_numpy(np.vstack([e.action for e in experiences if␣
↪e is not None])).long().to(device)
        rewards = torch.from_numpy(np.vstack([e.reward for e in experiences if␣
↪e is not None])).float().to(device)
        next_states = torch.from_numpy(np.vstack([e.next_state for e in␣
↪experiences if e is not None])).float().to(device)
        dones = torch.from_numpy(np.vstack([e.done for e in experiences if e is␣
↪not None]).astype(np.uint8)).float().to(device)

        return (states, actions, rewards, next_states, dones)
```

```
    def __len__(self):
        """Return the current size of internal memory."""
        return len(self.memory)
```

0.1.6 Define the Deep QLearning Agent

The Deep Q-learning update at iteration i uses the following loss function

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

where γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the Q-network parameters used to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters θ_i every C steps and are held fixed between individual updates.

```
[6]: class DQAgent():
    """Interacts with and learns from the environment."""

    def __init__(self, state_shape, action_space_size, seed):
        """Initialize an Agent object.

        Params
        ======
            state_shape (int): dimension of each state
            action_space_size (int): dimension of each action
            seed (int): random seed
        """
        self.state_shape = state_shape
        self.action_space_size = action_space_size
        self.seed = random.seed(seed)

        # Q-Network
        self.qnetwork_local = QNetwork(state_shape, action_space_size, seed).
↪to(device)
        self.qnetwork_target = QNetwork(state_shape, action_space_size, seed).
↪to(device)

        self.optimizer = optim.Adam(self.qnetwork_local.parameters(), lr=LR)

        # Replay memory
        self.memory = ReplayBuffer(BUFFER_SIZE, BATCH_SIZE, seed)

        # Initialize time step (for updating every UPDATE_EVERY steps)
        self.t_step = 0

    def step(self, state, action, reward, next_state, done):
```

```
        # Save experience in replay memory
        self.memory.add(state, action, reward, next_state, done)

        # Learn every UPDATE_EVERY time steps.
        self.t_step = (self.t_step + 1) % UPDATE_EVERY
        if self.t_step == 0:
            # If enough samples are available in memory, get random subset and␣
↪learn
            if len(self.memory) > BATCH_SIZE:
                experiences = self.memory.sample()
                self.learn(experiences, GAMMA)

    def act(self, state, eps=0.):
        """Returns actions for given state as per current policy.
        Params
        ======
            state (array_like): current state
            eps (float): epsilon, for epsilon-greedy action selection
        """
        # Epsilon-greedy action selection
        if random.random() > eps:
            state = torch.from_numpy(state).float().unsqueeze(0).to(device)
            action_values = self.qnetwork_local(state)
            return np.argmax(action_values.cpu().data.numpy())
        else:
            return random.choice(np.arange(self.action_space_size))

    def learn(self, experiences, gamma):
        """Update value parameters using given batch of experience tuples.

        Params
        ======
            experiences (Tuple[torch.Variable]): tuple of (s, a, r, s', done)␣
↪tuples
            gamma (float): discount factor
        """
        # Obtain random minibatch of tuples from D
        states, actions, rewards, next_states, dones = experiences

        ## Compute and minimize the loss
        ### Extract next maximum estimated value from target network
        q_targets_next = self.qnetwork_target(next_states).detach().max(1)[0].
↪unsqueeze(1)

        ### Calculate target value from bellman equation
        q_targets = rewards + gamma * q_targets_next * (1 - dones)
```

```
        ### Calculate expected value from local network
        q_expected = self.qnetwork_local(states).gather(1, actions)

        ### Loss calculation (we used Mean squared error)
        loss = F.mse_loss(q_expected, q_targets)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # ------------------- update target network ------------------- #
        self.soft_update(self.qnetwork_local, self.qnetwork_target, TAU)

    def soft_update(self, local_model, target_model, tau):
        """Soft update model parameters.
        _target =  *_local + (1 -  )*_target

        Params
        ======
            local_model (PyTorch model): weights will be copied from
            target_model (PyTorch model): weights will be copied to
            tau (float): interpolation parameter
        """
        for target_param, local_param in zip(target_model.parameters(),␣
↪local_model.parameters()):
            target_param.data.copy_(tau*local_param.data + (1.
↪0-tau)*target_param.data)
```

0.1.7 Define the Deep QLearning Agent Without Experience Replay

- you dont need a memory buffer to store the previous experiences
- the model is trained after each experience [state, action, reward, next_state, done],
- so the method step() of the agent launches the model training based only on the last experience instead of a batch of experiences

[17]:
```
class DQAgent_Without_ExperienceReplay():
    """Interacts with and learns from the environment."""

    def __init__(self, state_shape, action_space_size, seed):
        """Initialize an Agent object.

        Params
        ======
            state_shape (int): dimension of each state
            action_space_size (int): dimension of each action
            seed (int): random seed
        """
        self.state_shape = state_shape
```

```
        self.action_space_size = action_space_size
        self.seed = random.seed(seed)

        # Q-Network
        self.qnetwork_local = QNetwork(state_shape, action_space_size, seed).
↪to(device)
        self.qnetwork_target = QNetwork(state_shape, action_space_size, seed).
↪to(device)

        self.optimizer = optim.Adam(self.qnetwork_local.parameters(), lr=LR)



    def step(self, state, action, reward, next_state, done):
        # Transfor the current experience to tensor
        # launch the training based on the current experience
        self.learn(state, action, reward, next_state, done, GAMMA)


    def act(self, state, eps=0.):
        """Returns actions for given state as per current policy."""

        # Epsilon-greedy action selection
        if random.random() > eps:
            state = torch.from_numpy(state).float().unsqueeze(0).to(device)
            action_values = self.qnetwork_local(state)
            return np.argmax(action_values.cpu().data.numpy())
        else:
            return random.choice(np.arange(self.action_space_size))

    def learn(self, state, action, reward, next_state,done, gamma):
        """Update value parameters using given one experience tuple."""


        ## Compute and minimize the loss
        ### Extract next maximum estimated value from target network
        q_target_next = self.qnetwork_target(torch.tensor(next_state,␣
↪device=device)).detach().max().item()

        ### Calculate target value from bellman equation
        q_target = torch.tensor(reward, device=device) + gamma * q_target_next␣
↪* (1 - done)

        ### Calculate expected value from local network
        q_expected =  self.qnetwork_local(torch.tensor(state,␣
↪device=device))[action]
```

```
        ### Loss calculation (we used Mean squared error)
        loss = F.mse_loss(q_expected, q_target)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # ------------------- update target network ------------------- #
        self.soft_update(self.qnetwork_local, self.qnetwork_target, TAU)

    def soft_update(self, local_model, target_model, tau):

        for target_param, local_param in zip(target_model.parameters(),␣
↪local_model.parameters()):
            target_param.data.copy_(tau*local_param.data + (1.
↪0-tau)*target_param.data)
```

0.1.8 Training Process

The code for the training loop remains unchanged For both agents (DQN and DQN Without Experience Replay).

[8]:
```
def dqn(agent, n_episodes=1500, max_t=1000, eps_start=1.0, eps_end=0.01,␣
↪eps_decay=0.995):
    """Deep Q-Learning.

    Params
    ======
        n_episodes (int): maximum number of training episodes
        max_t (int): maximum number of timesteps per episode
        eps_start (float): starting value of epsilon, for epsilon-greedy action␣
↪selection
        eps_end (float): minimum value of epsilon
        eps_decay (float): multiplicative factor (per episode) for decreasing␣
↪epsilon
    """
    scores = []                        # list containing scores from each␣
↪episode
    scores_window = deque(maxlen=100)  # last 100 scores
    eps = eps_start                    # initialize epsilon

    for i_episode in range(1, n_episodes+1):
        state = env.reset()
        score = 0
        for t in range(max_t):
            action = agent.act(state, eps)
            next_state, reward, done, _ = env.step(action)
            agent.step(state, action, reward, next_state, done)
```

```
            state = next_state
            score += reward
            if done:
                break
        scores_window.append(score)       # save most recent score
        scores.append(score)              # save most recent score
        eps = max(eps_end, eps_decay*eps) # decrease epsilon

        print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.
↪mean(scores_window)), end="")
        if i_episode % 100 == 0:
            print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.
↪mean(scores_window)))

    torch.save(agent.qnetwork_local.state_dict(), 'checkpoint.pth')
    return scores
```

0.1.9 Creating the Gym environment CartPole-v1

This environment is part of the Classic Control environments which contains general information about the environment.

0.1.10 Description

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson in "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem". A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.

0.1.11 Action Space

The action is a ndarray with shape (1,) which can take values {0, 1} indicating the direction of the fixed force the cart is pushed with.

- 0: Push cart to the left
- 1: Push cart to the right

Note: The velocity that is reduced or increased by the applied force is not fixed and it depends on the angle the pole is pointing. The center of gravity of the pole varies the amount of energy needed to move the cart underneath it

0.1.12 Observation Space

The observation is a ndarray with shape (4,) with the values corresponding to the following positions and velocities:

- Cart Position
- Cart Velocity

- Pole Angle
- Pole Angular Velocity

0.1.13 Rewards

Since the goal is to keep the pole upright for as long as possible, a reward of +1 for every step taken, including the termination step, is allotted. The threshold for rewards is 500

[9]:
```
env = gym.make('CartPole-v1')
print(env.reset())
print('State shape: ', env.observation_space.shape[0])
print('Number of actions: ', env.action_space.n)
```

```
[-0.02901609  0.01683903  0.04125848  0.02246835]
State shape:  4
Number of actions:  2
```

```
/usr/local/lib/python3.10/dist-packages/gym/core.py:317: DeprecationWarning:
WARN: Initializing wrapper in old step API which returns one bool instead

of two. It is recommended to set `new_step_api=True` to use new step API. This

will be the default behaviour in future.
  deprecation(
/usr/local/lib/python3.10/dist-
packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning:
WARN: Initializing environment in old step API which returns one bool

instead of two. It is recommended to set `new_step_api=True` to use new step

API. This will be the default behaviour in future.
  deprecation(
```

[10]:
```
load_dqn = True
```

[11]:
```
DQagent = DQAgent(state_shape=env.observation_space.shape[0],␣
 ↪action_space_size=env.action_space.n, seed=0)
if not load_dqn:
    scores_dqn = dqn(DQagent)
    torch.save(DQagent.qnetwork_local.state_dict(), "/content/
 ↪qnetwork_local-DQAgent.pth")
    torch.save(DQagent.qnetwork_target.state_dict(), "/content/
 ↪qnetwork_target-DQAgent.pth")
else:
    DQagent.qnetwork_local.load_state_dict(torch.load("/content/
 ↪qnetwork_local-DQAgent.pth"))
    DQagent.qnetwork_target.load_state_dict(torch.load("/content/
 ↪qnetwork_target-DQAgent.pth"))
```

[18]:

```
DQagent_Without_ExperienceReplay =␣
 ↪DQAgent_Without_ExperienceReplay(state_shape=env.observation_space.shape[0],␣
 ↪action_space_size=env.action_space.n, seed=0)
scores_dqn_without_ExperienceReplay = dqn(DQagent_Without_ExperienceReplay)
torch.save(DQagent_Without_ExperienceReplay.qnetwork_local.state_dict(), "/
 ↪content/qnetwork_local-DQagent_Without_ExperienceReplay.pth")
torch.save(DQagent_Without_ExperienceReplay.qnetwork_target.state_dict(), "/
 ↪content/qnetwork_target-DQagent_Without_ExperienceReplay.pth")
```

```
Episode 100     Average Score: 20.96
Episode 200     Average Score: 19.29
Episode 300     Average Score: 18.75
Episode 400     Average Score: 20.23
Episode 500     Average Score: 18.18
Episode 600     Average Score: 14.88
Episode 700     Average Score: 9.43
Episode 800     Average Score: 10.11
Episode 900     Average Score: 9.44
Episode 1000    Average Score: 9.71
Episode 1100    Average Score: 28.34
Episode 1200    Average Score: 38.08
Episode 1300    Average Score: 9.51
Episode 1400    Average Score: 43.38
Episode 1500    Average Score: 9.44
```

0.2 Comparing DQN and DQN-Without_ExperienceReplay

Plotting the time series of scores (scores_dqn & scores_dqn_without_ExperienceReplay) I can use Pandas to quickly plot the time series of scores along with a 100 episode moving average. Note that training stops as soon as the rolling average crosses the target score.

[14]:

```
#scores_dqn    = pd.Series(scores_dqn , name="scores")
#scores_dqn .describe()
```

[19]:

```
scores_dqn_without_ExperienceReplay   = pd.
 ↪Series(scores_dqn_without_ExperienceReplay, name="scores")
scores_dqn_without_ExperienceReplay.describe()
```

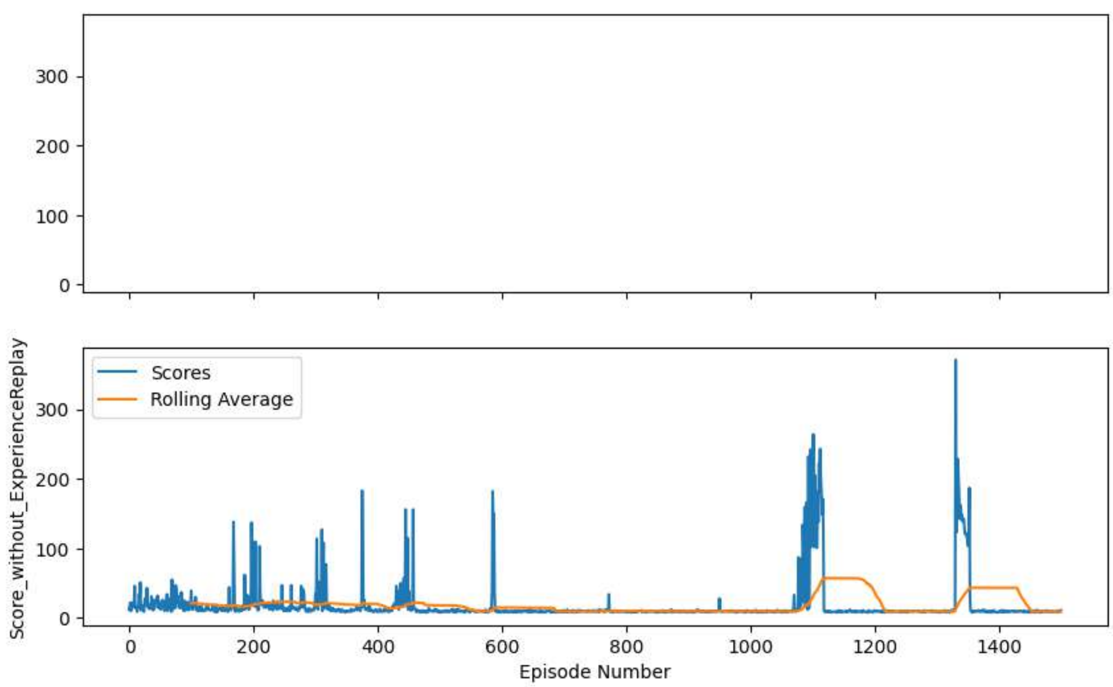
[19]:

```
count    1500.000000
mean       18.648667
std        31.997789
min         8.000000
25%         9.000000
50%        10.000000
75%        12.000000
max       371.000000
```

```
Name: scores, dtype: float64
```

[20]:

```
fig, ax = plt.subplots(2, 1, figsize=(10, 6), sharex=True, sharey=True)
# _ = scores_dqn.plot(ax=ax[0], label="Scores")
# _ = (scores_dqn.rolling(window=100)
#             .mean()
#             .rename("Rolling Average")
#             .plot(ax=ax[0]))
# _=ax[0].legend()
# _=ax[0].set_ylabel("DQN Score")

_ = scores_dqn_without_ExperienceReplay.plot(ax=ax[1], label="Scores")
_ = (scores_dqn_without_ExperienceReplay.rolling(window=100)
           .mean()
           .rename("Rolling Average")
           .plot(ax=ax[1]))
_=ax[1].legend()
_ = ax[1].set_ylabel("Score_without_ExperienceReplay")
_ = ax[1].set_xlabel("Episode Number")
```



szj7nhtum

December 17, 2023

0.1 ## Double Deep Q-Network (DDQN)

In this notebook, you will implement a Double DQN agent with OpenAI Gym's CartPole-v1 environment.

0.1.1 Import the Necessary Packages

[ ]:
```
import gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from tqdm import tqdm
import imageio
from collections import deque, namedtuple

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

0.1.2 Define some hyperparameter

[ ]:
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

0.1.3 Define Neural Network Architecture.

Since `CartPole-v1` environment is sort of simple envs, we don't need complicated architecture. We just need non-linear function approximator that maps from state to action.

[ ]:
```
class QNetwork(nn.Module):

    def __init__(self, state_shape, action_space_size, seed):
        """Initialize parameters and build model.
        Params
        ======
            state_shape (int): Dimension of each state
            action_space_size (int): Dimension of each action
            seed (int): Random seed
        """
        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_shape, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, action_space_size)

    def forward(self, state):
        """Build a network that maps state -> action values."""
        x = self.fc1(state)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x=self.fc3(x)
        return x
```

0.1.4 Define Replay Buffer

0.1.5 Experience Replay

To perform *experience replay* the authors store the agent's experiences e_t as represented by the tuple

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

consisting of the observed state in period t, the reward received in period t, the action taken in period t, and the resulting state in period $t+1$. The dataset of agent experiences at period t consists of the set of past experiences.

$$D_t = \{e1, e2, ..., e_t\}$$

Depending on the task it may note be feasible for the agent to store the entire history of past experiences.

During learning Q-learning updates are computed based on samples (or minibatches) of experience (s, a, r, s'), drawn uniformly at random from the pool of stored samples D_t.

The following is my Python implmentation of these ideas.

[ ]:
```
class ReplayBuffer:
    """Fixed-size buffer to store experience tuples."""

    def __init__(self, buffer_size, batch_size, seed):
        """Initialize a ReplayBuffer object.

        Params
        ======
            buffer_size (int): maximum size of buffer
            batch_size (int): size of each training batch
            seed (int): random seed
        """
        self.memory = deque(maxlen=buffer_size)
        self.batch_size = batch_size
        self.experience = namedtuple("Experience", field_names=["state", "action", "reward", "next_state", "done"])
        self.seed = random.seed(seed)

    def add(self, state, action, reward, next_state, done):
        """Add a new experience to memory."""
        e = self.experience(state, action, reward, next_state, done)
        self.memory.append(e)

    def sample(self):
        """Randomly sample a batch of experiences from memory."""
        experiences = random.sample(self.memory, k=self.batch_size)

        states = torch.from_numpy(np.vstack([e.state for e in experiences if e is not None])).float().to(device)
        actions = torch.from_numpy(np.vstack([e.action for e in experiences if e is not None])).long().to(device)
        rewards = torch.from_numpy(np.vstack([e.reward for e in experiences if e is not None])).float().to(device)
        next_states = torch.from_numpy(np.vstack([e.next_state for e in experiences if e is not None])).float().to(device)
        dones = torch.from_numpy(np.vstack([e.done for e in experiences if e is not None]).astype(np.uint8)).float().to(device)

        return (states, actions, rewards, next_states, dones)

    def __len__(self):
        """Return the current size of internal memory."""
```

```
        return len(self.memory)
```

1 Improving the DQN Agent using Double Q-Learning

The key idea behind Double Q-learning is to reduce overestimations of Q-values by separating the selection of actions from the evaluation of those actions so that a different Q-network can be used in each step. When applying Double Q-learning to extend the DQN algorithm one can use the online Q-network, $Q(S, a; \theta)$, to select the actions and then the target Q-network, $Q(S, a; \theta^-)$, to evaluate the selected actions.

$$Y_t^{DoubleDQN} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \theta_t), \theta_t^-)$$

[ ]:
```
class DDQAgent():
    """Interacts with and learns from the environment."""

    def __init__(self, state_shape, action_space_size, seed):
        """Initialize an Agent object.

        Params
        ======
            state_shape (int): dimension of each state
            action_space_size (int): dimension of each action
            seed (int): random seed
        """
        self.state_shape = state_shape
        self.action_space_size = action_space_size
        self.seed = random.seed(seed)

        # Q-Network
        self.qnetwork_local = QNetwork(state_shape, action_space_size, seed).
 ↪to(device)
        self.qnetwork_target = QNetwork(state_shape, action_space_size, seed).
 ↪to(device)

        self.optimizer = optim.Adam(self.qnetwork_local.parameters(), lr=LR)

        # Replay memory
        self.memory = ReplayBuffer(BUFFER_SIZE, BATCH_SIZE, seed)

        # Initialize time step (for updating every UPDATE_EVERY steps)
        self.t_step = 0

    def step(self, state, action, reward, next_state, done):
        # Save experience in replay memory
        self.memory.add(state, action, reward, next_state, done)
```

```
        # Learn every UPDATE_EVERY time steps.
        self.t_step = (self.t_step + 1) % UPDATE_EVERY
        if self.t_step == 0:
            # If enough samples are available in memory, get random subset and␣
↪learn
            if len(self.memory) > BATCH_SIZE:
                experiences = self.memory.sample()
                self.learn(experiences, GAMMA)

    def act(self, state, eps=0.):
        """Returns actions for given state as per current policy.
        Params
        ======
            state (array_like): current state
            eps (float): epsilon, for epsilon-greedy action selection
        """
        # Epsilon-greedy action selection
        if random.random() > eps:
            state = torch.from_numpy(state).float().unsqueeze(0).to(device)
            action_values = self.qnetwork_local(state)
            return np.argmax(action_values.cpu().data.numpy())
        else:
            return random.choice(np.arange(self.action_space_size))

    def learn(self, experiences, gamma):
        """Update value parameters using given batch of experience tuples.

        Params
        ======
            experiences (Tuple[torch.Variable]): tuple of (s, a, r, s', done)␣
↪tuples
            gamma (float): discount factor
        """
        # Obtain random minibatch of tuples from D
        states, actions, rewards, next_states, dones = experiences

        ## Compute and minimize the loss
        """Select the greedy action for the next_state given Local Q-network."""
        # q = self.qnetwork_local(next_states).detach()
        # print(q, q.shape)
        actions_for_next_states = self.qnetwork_local(next_states).argmax(1)

        """Compute the next_Q-values by evaluating the actions given the␣
↪next_states and the Target Q-network."""
        # q = self.qnetwork_target(next_states)
        # print(q.shape, actions_for_next_states.shape, actions.shape)
```

```
        q_targets_next = self.qnetwork_target(next_states).gather(1,␣
↪actions_for_next_states.view(-1, 1))

        ### Calculate target value from bellman equation
        q_targets = rewards + gamma * q_targets_next * (1 - dones)

        ### Calculate expected value from local network
        q_expected = self.qnetwork_local(states).gather(1, actions)

        ### Loss calculation (we used Mean squared error)
        loss = F.mse_loss(q_expected, q_targets)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # ------------------- update target network ------------------- #
        self.soft_update(self.qnetwork_local, self.qnetwork_target, TAU)

    def soft_update(self, local_model, target_model, tau):

        for target_param, local_param in zip(target_model.parameters(),␣
↪local_model.parameters()):
            target_param.data.copy_(tau*local_param.data + (1.
↪0-tau)*target_param.data)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

1.0.1 Define the Deep QLearning Network Agent

Before Training the Double DQN Agent, we re-implement the DQN Agent in order to compare them

[ ]:
```
class DQAgent():
    """Interacts with and learns from the environment."""

    def __init__(self, state_shape, action_space_size, seed):

        self.state_shape = state_shape
        self.action_space_size = action_space_size
        self.seed = random.seed(seed)

        # Q-Network
```

```
        self.qnetwork_local = QNetwork(state_shape, action_space_size, seed).
↪to(device)
        self.qnetwork_target = QNetwork(state_shape, action_space_size, seed).
↪to(device)

        self.optimizer = optim.Adam(self.qnetwork_local.parameters(), lr=LR)

        # Replay memory
        self.memory = ReplayBuffer(BUFFER_SIZE, BATCH_SIZE, seed)

        # Initialize time step (for updating every UPDATE_EVERY steps)
        self.t_step = 0

    def step(self, state, action, reward, next_state, done):
        # Save experience in replay memory
        self.memory.add(state, action, reward, next_state, done)

        # Learn every UPDATE_EVERY time steps.
        self.t_step = (self.t_step + 1) % UPDATE_EVERY
        if self.t_step == 0:
            # If enough samples are available in memory, get random subset and␣
↪learn
            if len(self.memory) > BATCH_SIZE:
                experiences = self.memory.sample()
                self.learn(experiences, GAMMA)

    def act(self, state, eps=0.):

        # Epsilon-greedy action selection
        if random.random() > eps:
            state = torch.from_numpy(state).float().unsqueeze(0).to(device)
            action_values = self.qnetwork_local(state)
            return np.argmax(action_values.cpu().data.numpy())
        else:
            return random.choice(np.arange(self.action_space_size))

    def learn(self, experiences, gamma):

        # Obtain random minibatch of tuples from D
        states, actions, rewards, next_states, dones = experiences

        ## Compute and minimize the loss
        ### Extract next maximum estimated value from target network
        q_targets_next = self.qnetwork_target(next_states).detach().max(1)[0].
↪unsqueeze(1)

        ### Calculate target value from bellman equation
```

```
        q_targets = rewards + gamma * q_targets_next * (1 - dones)

        ### Calculate expected value from local network
        q_expected = self.qnetwork_local(states).gather(1, actions)

        ### Loss calculation (we used Mean squared error)
        loss = F.mse_loss(q_expected, q_targets)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # ------------------- update target network ------------------- #
        self.soft_update(self.qnetwork_local, self.qnetwork_target, TAU)

    def soft_update(self, local_model, target_model, tau):

        for target_param, local_param in zip(target_model.parameters(),␣
↪local_model.parameters()):
            target_param.data.copy_(tau*local_param.data + (1.
↪0-tau)*target_param.data)
```

1.0.2 Training Process

The code for the training loop remains unchanged For both agents (Double-DQN and DQN).

[ ]:
```
def dqn(agent, n_episodes=1700, max_t=1000, eps_start=1.0, eps_end=0.01,␣
↪eps_decay=0.995):

    scores = []                        # list containing scores from each␣
↪episode
    scores_window = deque(maxlen=100)  # last 100 scores
    eps = eps_start                    # initialize epsilon

    for i_episode in range(1, n_episodes+1):
        state = env.reset()
        score = 0
        for t in range(max_t):
            action = agent.act(state, eps)
            next_state, reward, done, _ = env.step(action)
            agent.step(state, action, reward, next_state, done)
            state = next_state
            score += reward
            if done:
                break
        scores_window.append(score)       # save most recent score
        scores.append(score)              # save most recent score
        eps = max(eps_end, eps_decay*eps) # decrease epsilon
```

```
        print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.
↪mean(scores_window)), end="")
        if i_episode % 100 == 0:
            print('\rEpisode {}\tAverage Score: {:.2f}'.format(i_episode, np.
↪mean(scores_window)))

    torch.save(agent.qnetwork_local.state_dict(), 'checkpoint.pth')
    return scores
```

1.0.3 Creating the Gym environment CartPole-v1

This environment is part of the Classic Control environments which contains general information about the environment.

1.0.4 Description

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson in "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem". A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.

1.0.5 Action Space

The action is a ndarray with shape (1,) which can take values {0, 1} indicating the direction of the fixed force the cart is pushed with.

- 0: Push cart to the left
- 1: Push cart to the right

Note: The velocity that is reduced or increased by the applied force is not fixed and it depends on the angle the pole is pointing. The center of gravity of the pole varies the amount of energy needed to move the cart underneath it

1.0.6 Observation Space

The observation is a ndarray with shape (4,) with the values corresponding to the following positions and velocities:

- Cart Position
- Cart Velocity
- Pole Angle
- Pole Angular Velocity

1.0.7 Rewards

Since the goal is to keep the pole upright for as long as possible, a reward of +1 for every step taken, including the termination step, is allotted. The threshold for rewards is 500

```
[ ]: env = gym.make('CartPole-v1')
     print(env.reset())
     print('State shape: ', env.observation_space.shape[0])
     print('Number of actions: ', env.action_space.n)
```

```
[ 0.02899956 -0.01435325  0.00254188  0.02645365]
State shape:  4
Number of actions:  2

/usr/local/lib/python3.10/dist-packages/gym/core.py:317: DeprecationWarning:
WARN: Initializing wrapper in old step API which returns one bool instead

of two. It is recommended to set `new_step_api=True` to use new step API. This

will be the default behaviour in future.
  deprecation(
/usr/local/lib/python3.10/dist-
packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning:
WARN: Initializing environment in old step API which returns one bool

instead of two. It is recommended to set `new_step_api=True` to use new step

API. This will be the default behaviour in future.
  deprecation(
```

1.0.8 Training the DDQagent using Double DQN

```
[ ]: DDQagent = DDQAgent(state_shape=env.observation_space.shape[0],
       ↪action_space_size=env.action_space.n, seed=0)
     scores_DDQAgent = dqn(DDQagent)
```

```
Episode 100     Average Score: 18.71
Episode 200     Average Score: 11.94
Episode 300     Average Score: 11.17
Episode 400     Average Score: 10.15
Episode 500     Average Score: 9.96
Episode 600     Average Score: 9.69
Episode 700     Average Score: 11.22
Episode 800     Average Score: 12.59
Episode 900     Average Score: 29.05
Episode 1000    Average Score: 192.20
Episode 1100    Average Score: 239.04
Episode 1200    Average Score: 222.89
Episode 1300    Average Score: 216.29
Episode 1400    Average Score: 225.62
Episode 1500    Average Score: 262.36
Episode 1600    Average Score: 272.68
Episode 1700    Average Score: 253.70
```

1.0.9 Training the DQagent using DQN

[ ]:
```
DQagent = DQAgent(state_shape=env.observation_space.shape[0],␣
↪action_space_size=env.action_space.n, seed=0)
scores_DQAgent = dqn(DQagent)
```

```
Episode 100     Average Score: 16.41
Episode 200     Average Score: 12.13
Episode 300     Average Score: 11.09
Episode 400     Average Score: 10.27
Episode 500     Average Score: 10.08
Episode 600     Average Score: 10.01
Episode 700     Average Score: 11.12
Episode 800     Average Score: 14.18
Episode 900     Average Score: 31.41
Episode 1000    Average Score: 155.78
Episode 1100    Average Score: 170.00
Episode 1200    Average Score: 185.71
Episode 1300    Average Score: 304.15
Episode 1400    Average Score: 294.68
Episode 1500    Average Score: 153.37
Episode 1600    Average Score: 147.27
Episode 1700    Average Score: 135.29
```

1.1 Comparing Double DQN and DQN

Plotting the time series of scores (scores_DDQAgent & scores_DQAgent) I can use Pandas to quickly plot the time series of scores along with a 100 episode moving average. Note that training stops as soon as the rolling average crosses the target score.

[ ]:
```
scores_DDQAgent= pd.Series(scores_DDQAgent, name="scores_DDQAgent")
scores_DDQAgent.describe()
```

[ ]:
```
count    1700.000000
mean      118.191765
std       117.236603
min         8.000000
25%        10.000000
50%        26.000000
75%       222.000000
max       500.000000
Name: scores_DDQAgent, dtype: float64
```

[ ]:
```
scores_DQAgent= pd.Series(scores_DQAgent, name="scores_DQAgent")
scores_DQAgent.describe()
```

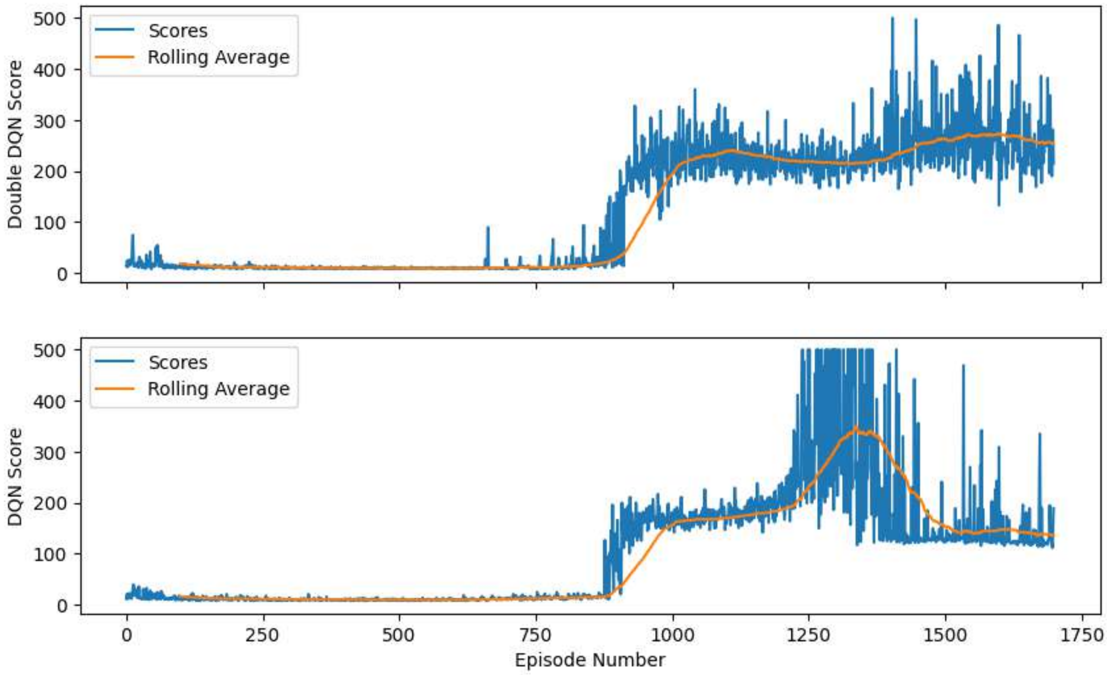
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
```

```
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:
```
count    1700.000000
mean       98.408824
std       110.992427
min         8.000000
25%        11.000000
50%        21.000000
75%       163.250000
max       500.000000
Name: scores_DQAgent, dtype: float64
```

[ ]:
```python
fig, ax = plt.subplots(2, 1, figsize=(10, 6), sharex=True, sharey=True)
_ = scores_DDQAgent.plot(ax=ax[0], label="Scores")
_ = (scores_DDQAgent.rolling(window=100)
            .mean()
            .rename("Rolling Average")
            .plot(ax=ax[0]))
_=ax[0].legend()
_=ax[0].set_ylabel("Double DQN Score")

_ = scores_DQAgent.plot(ax=ax[1], label="Scores")
_ = (scores_DQAgent.rolling(window=100)
            .mean()
            .rename("Rolling Average")
            .plot(ax=ax[1]))
_=ax[1].legend()
_ = ax[1].set_ylabel("DQN Score")
_ = ax[1].set_xlabel("Episode Number")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```


Scores
Rolling Average
Double DQN Score
0
100
200
300
400
500
Scores
Rolling Average
DQN Score
0
250
500
750
1000
1250
1500
1750
Episode Number

elddkd0vb

December 17, 2023

[1]:
```
!pip install -r requirements_lab1.txt
```

```
Requirement already satisfied: numpy in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from -r
requirements_lab1.txt (line 1)) (1.24.3)
Requirement already satisfied: gymnasium in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from -r
requirements_lab1.txt (line 2)) (0.29.1)
Requirement already satisfied: pygame in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from -r
requirements_lab1.txt (line 3)) (2.5.2)
Requirement already satisfied: jupyter in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from -r
requirements_lab1.txt (line 4)) (1.0.0)
Requirement already satisfied: pandas in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from -r
requirements_lab1.txt (line 5)) (1.5.3)
Requirement already satisfied: matplotlib in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from -r
requirements_lab1.txt (line 6)) (3.7.1)
Requirement already satisfied: tqdm in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from -r
requirements_lab1.txt (line 7)) (4.65.0)
Requirement already satisfied: imageio in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from -r
requirements_lab1.txt (line 8)) (2.31.1)
Requirement already satisfied: cloudpickle>=1.2.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
gymnasium->-r requirements_lab1.txt (line 2)) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
gymnasium->-r requirements_lab1.txt (line 2)) (4.6.3)
Requirement already satisfied: farama-notifications>=0.0.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
gymnasium->-r requirements_lab1.txt (line 2)) (0.0.4)
Requirement already satisfied: notebook in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter->-r
requirements_lab1.txt (line 4)) (6.5.4)
```

```
Requirement already satisfied: qtconsole in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter->-r
requirements_lab1.txt (line 4)) (5.4.2)
Requirement already satisfied: jupyter-console in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter->-r
requirements_lab1.txt (line 4)) (6.6.3)
Requirement already satisfied: nbconvert in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter->-r
requirements_lab1.txt (line 4)) (6.5.4)
Requirement already satisfied: ipykernel in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter->-r
requirements_lab1.txt (line 4)) (6.19.2)
Requirement already satisfied: ipywidgets in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter->-r
requirements_lab1.txt (line 4)) (8.0.4)
Requirement already satisfied: python-dateutil>=2.8.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from pandas->-r
requirements_lab1.txt (line 5)) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from pandas->-r
requirements_lab1.txt (line 5)) (2022.7)
Requirement already satisfied: contourpy>=1.0.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
matplotlib->-r requirements_lab1.txt (line 6)) (1.0.5)
Requirement already satisfied: cycler>=0.10 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
matplotlib->-r requirements_lab1.txt (line 6)) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
matplotlib->-r requirements_lab1.txt (line 6)) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
matplotlib->-r requirements_lab1.txt (line 6)) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
matplotlib->-r requirements_lab1.txt (line 6)) (23.0)
Requirement already satisfied: pillow>=6.2.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
matplotlib->-r requirements_lab1.txt (line 6)) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
matplotlib->-r requirements_lab1.txt (line 6)) (3.0.9)
Requirement already satisfied: imageio-ffmpeg in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from imageio->-r
requirements_lab1.txt (line 8)) (0.4.9)
Requirement already satisfied: psutil in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from imageio->-r
requirements_lab1.txt (line 8)) (5.9.0)
```

```
Requirement already satisfied: av in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from imageio->-r
requirements_lab1.txt (line 8)) (10.0.0)
Requirement already satisfied: six>=1.5 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from python-
dateutil>=2.8.1->pandas->-r requirements_lab1.txt (line 5)) (1.16.0)
Requirement already satisfied: setuptools in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from imageio-
ffmpeg->imageio->-r requirements_lab1.txt (line 8)) (67.8.0)
Requirement already satisfied: appnope in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (0.1.2)
Requirement already satisfied: comm>=0.1.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (0.1.2)
Requirement already satisfied: debugpy>=1.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (1.5.1)
Requirement already satisfied: ipython>=7.23.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (8.12.0)
Requirement already satisfied: jupyter-client>=6.1.12 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (8.1.0)
Requirement already satisfied: matplotlib-inline>=0.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (0.1.6)
Requirement already satisfied: nest-asyncio in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (1.5.6)
Requirement already satisfied: pyzmq>=17 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (25.1.0)
Requirement already satisfied: tornado>=6.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (6.2)
Requirement already satisfied: traitlets>=5.4.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (5.7.1)
Requirement already satisfied: widgetsnbextension~=4.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipywidgets->jupyter->-r requirements_lab1.txt (line 4)) (4.0.5)
Requirement already satisfied: jupyterlab-widgets~=3.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipywidgets->jupyter->-r requirements_lab1.txt (line 4)) (3.0.5)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
console->jupyter->-r requirements_lab1.txt (line 4)) (5.3.0)
```

```
Requirement already satisfied: prompt-toolkit>=3.0.30 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
console->jupyter->-r requirements_lab1.txt (line 4)) (3.0.36)
Requirement already satisfied: pygments in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
console->jupyter->-r requirements_lab1.txt (line 4)) (2.15.1)
Requirement already satisfied: lxml in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (4.9.2)
Requirement already satisfied: beautifulsoup4 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (4.12.2)
Requirement already satisfied: bleach in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (4.1.0)
Requirement already satisfied: defusedxml in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (0.4)
Requirement already satisfied: jinja2>=3.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (3.1.2)
Requirement already satisfied: jupyterlab-pygments in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (0.1.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (2.1.1)
Requirement already satisfied: mistune<2,>=0.8.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (0.5.13)
Requirement already satisfied: nbformat>=5.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (5.7.0)
Requirement already satisfied: pandocfilters>=1.4.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (1.5.0)
Requirement already satisfied: tinycss2 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (1.2.1)
Requirement already satisfied: argon2-cffi in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
notebook->jupyter->-r requirements_lab1.txt (line 4)) (21.3.0)
```

```
Requirement already satisfied: ipython-genutils in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
notebook->jupyter->-r requirements_lab1.txt (line 4)) (0.2.0)
Requirement already satisfied: Send2Trash>=1.8.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
notebook->jupyter->-r requirements_lab1.txt (line 4)) (1.8.0)
Requirement already satisfied: terminado>=0.8.3 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
notebook->jupyter->-r requirements_lab1.txt (line 4)) (0.17.1)
Requirement already satisfied: prometheus-client in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
notebook->jupyter->-r requirements_lab1.txt (line 4)) (0.14.1)
Requirement already satisfied: nbclassic>=0.4.7 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
notebook->jupyter->-r requirements_lab1.txt (line 4)) (0.5.5)
Requirement already satisfied: qtpy>=2.0.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
qtconsole->jupyter->-r requirements_lab1.txt (line 4)) (2.2.0)
Requirement already satisfied: backcall in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (0.2.0)
Requirement already satisfied: decorator in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (5.1.1)
Requirement already satisfied: jedi>=0.16 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (0.18.1)
Requirement already satisfied: pickleshare in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (0.7.5)
Requirement already satisfied: stack-data in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (0.2.0)
Requirement already satisfied: pexpect>4.3 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line 4)) (4.8.0)
Requirement already satisfied: platformdirs>=2.5 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
core!=5.0.*,>=4.12->jupyter-console->jupyter->-r requirements_lab1.txt (line 4))
(3.9.1)
Requirement already satisfied: jupyter-server>=1.8 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbclassic>=0.4.7->notebook->jupyter->-r requirements_lab1.txt (line 4)) (2.5.0)
Requirement already satisfied: notebook-shim>=0.1.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbclassic>=0.4.7->notebook->jupyter->-r requirements_lab1.txt (line 4)) (0.2.2)
Requirement already satisfied: fastjsonschema in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
```

```
nbformat>=5.1->nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (2.16.2)
Requirement already satisfied: jsonschema>=2.6 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
nbformat>=5.1->nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (4.17.3)
Requirement already satisfied: wcwidth in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from prompt-
toolkit>=3.0.30->jupyter-console->jupyter->-r requirements_lab1.txt (line 4))
(0.2.5)
Requirement already satisfied: ptyprocess in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
terminado>=0.8.3->notebook->jupyter->-r requirements_lab1.txt (line 4)) (0.7.0)
Requirement already satisfied: argon2-cffi-bindings in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
argon2-cffi->notebook->jupyter->-r requirements_lab1.txt (line 4)) (21.2.0)
Requirement already satisfied: soupsieve>1.2 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
beautifulsoup4->nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (2.4)
Requirement already satisfied: webencodings in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
bleach->nbconvert->jupyter->-r requirements_lab1.txt (line 4)) (0.5.1)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
jedi>=0.16->ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line
4)) (0.8.3)
Requirement already satisfied: attrs>=17.4.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r requirements_lab1.txt
(line 4)) (22.1.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r requirements_lab1.txt
(line 4)) (0.18.0)
Requirement already satisfied: anyio>=3.1.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r requirements_lab1.txt (line
4)) (3.5.0)
Requirement already satisfied: jupyter-events>=0.4.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r requirements_lab1.txt (line
4)) (0.6.3)
Requirement already satisfied: jupyter-server-terminals in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r requirements_lab1.txt (line
4)) (0.4.4)
Requirement already satisfied: websocket-client in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r requirements_lab1.txt (line
4)) (0.58.0)
```

```
Requirement already satisfied: cffi>=1.0.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
argon2-cffi-bindings->argon2-cffi->notebook->jupyter->-r requirements_lab1.txt
(line 4)) (1.15.1)
Requirement already satisfied: executing in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from stack-
data->ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line 4))
(0.8.3)
Requirement already satisfied: asttokens in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from stack-
data->ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line 4))
(2.0.5)
Requirement already satisfied: pure-eval in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from stack-
data->ipython>=7.23.1->ipykernel->jupyter->-r requirements_lab1.txt (line 4))
(0.2.2)
Requirement already satisfied: idna>=2.8 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
anyio>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
requirements_lab1.txt (line 4)) (3.4)
Requirement already satisfied: sniffio>=1.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
anyio>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
requirements_lab1.txt (line 4)) (1.2.0)
Requirement already satisfied: pycparser in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook->jupyter->-r
requirements_lab1.txt (line 4)) (2.21)
Requirement already satisfied: python-json-logger>=2.0.4 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
events>=0.4.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
requirements_lab1.txt (line 4)) (2.0.7)
Requirement already satisfied: pyyaml>=5.3 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
events>=0.4.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
requirements_lab1.txt (line 4)) (6.0)
Requirement already satisfied: rfc3339-validator in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
events>=0.4.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
requirements_lab1.txt (line 4)) (0.1.4)
Requirement already satisfied: rfc3986-validator>=0.1.1 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from jupyter-
events>=0.4.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook->jupyter->-r
requirements_lab1.txt (line 4)) (0.1.1)
Requirement already satisfied: fqdn in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r requirements_lab1.txt
(line 4)) (1.5.1)
```

```
Requirement already satisfied: isoduration in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r requirements_lab1.txt
(line 4)) (20.11.0)
Requirement already satisfied: jsonpointer>1.13 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r requirements_lab1.txt
(line 4)) (2.1)
Requirement already satisfied: uri-template in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r requirements_lab1.txt
(line 4)) (1.3.0)
Requirement already satisfied: webcolors>=1.11 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r requirements_lab1.txt
(line 4)) (1.13)
Requirement already satisfied: arrow>=0.15.0 in
/Users/abdelkrimzitouni/anaconda3/lib/python3.11/site-packages (from
isoduration->jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter->-r
requirements_lab1.txt (line 4)) (1.2.3)
```

Before you solve a Reinforcement Learning problem you need to define what are

- the environment
- the states
- the actions
- the rewards

We are using the `Taxi-v3` environment from OpenAI's gym: https://gymnasium.farama.org/environments/toy_text/taxi/

`Taxi-v3` is an easy environment because the action space is small, and the state space is large but finite.

Environments with a finite number of actions and states are called tabular

0.0.1 Import the Gymnasium Library

[2]:
```python
import gymnasium as gym
```

0.0.2 We create an environment with gym.make()

[3]:
```python
env=gym.make("Taxi-v3",render_mode="rgb_array")
```

0.0.3 We reset the environment to its initial state with state = env.reset()

[4]:
```
state=env.reset()
```

State space There are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is in the taxi), and 4 destination locations.

Destination on the map are represented with the first letter of the color.

Passenger locations:

0: Red

1: Green

2: Yellow

3: Blue

4: In taxi

Destinations:

0: Red

1: Green

2: Yellow

3: Blue

An observation is returned as an int() that encodes the corresponding state, calculated by ((taxi_row * 5 + taxi_col) * 5 + passenger_location) * 4 + destination

Note that there are 400 states that can actually be reached during an episode. The missing states correspond to situations in which the passenger is at the same location as their destination, as this typically signals the end of an episode. Four additional states can be observed right after a successful episodes, when both the passenger and the taxi are at the destination. This gives a total of 404 reachable discrete states.

[19]:
```
print("State Space {}".format(env.observation_space.n))
```

```
State Space 500
```

Action space The action shape is (1,) in the range {0, 5} indicating which direction to move the taxi or to pickup/drop off passengers.

0: Move south (down)

1: Move north (up)

2: Move east (right)

3: Move west (left)

4: Pickup passenger

5: Drop off passenger

[6]:
```
print("Action Space {}".format(env.action_space.n))
```

```
Action Space 6
```

0.0.4 Rewards

- -1 per step unless other reward is triggered.
- +20 delivering passenger.
- -10 executing "pickup" and "drop-off" actions illegally.

An action that results a noop, like moving into a wall, will incur the time step penalty. Noops can be avoided by sampling the action_mask returned in info.

0.0.5 Episode End

The episode ends if the following happens:

- Termination: 1. The taxi drops off the passenger.
- Truncation (when using the time_limit wrapper): 1. The length of the episode is 200.

0.0.6 Information

step() and reset() return a dict with the following keys:

- p - transition proability for the state.
- action_mask - if actions will cause a transition to a new state.

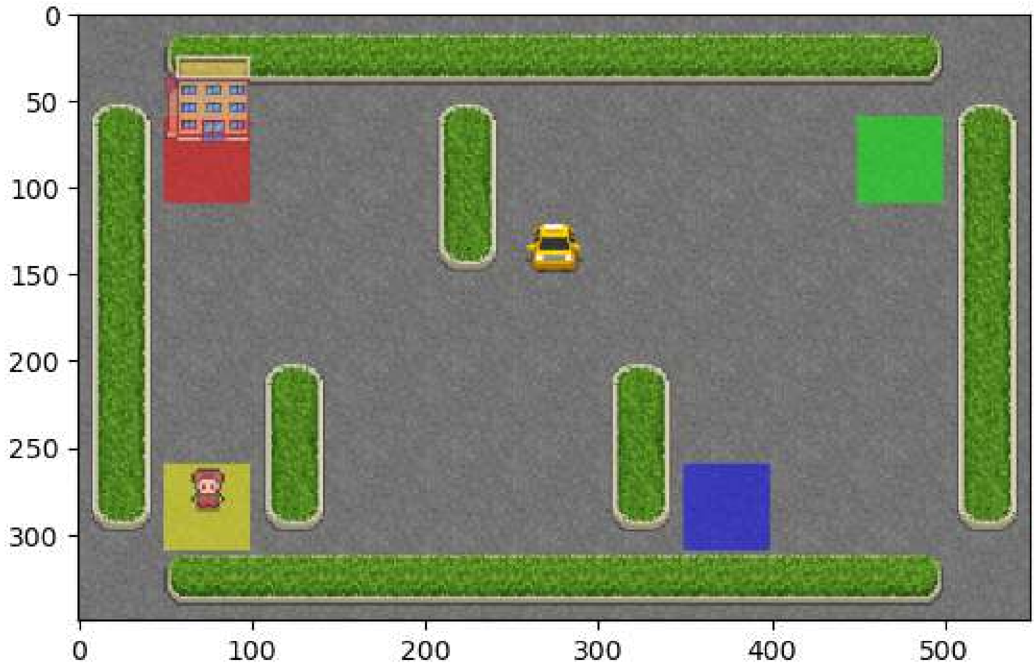
As taxi is not stochastic, the transition probability is always 1.0. Implementing a transitional probability in line with the Dietterich paper ('The fickle taxi task') is a TODO.

For some cases, taking an action will have no effect on the state of the episode. In v0.25.0, info["action_mask"] contains a np.ndarray for each of the actions specifying if the action will change the state.

To sample a modifying action, use action = env.action_space.sample(info["action_mask"]) Or with a Q-value based algorithm action = np.argmax(q_values[obs, np.where(info["action_mask"] == 1)[0]]).

[7]:
```
import matplotlib.pyplot as plt
env.reset()

image=env.render()
plt.imshow(image)
```

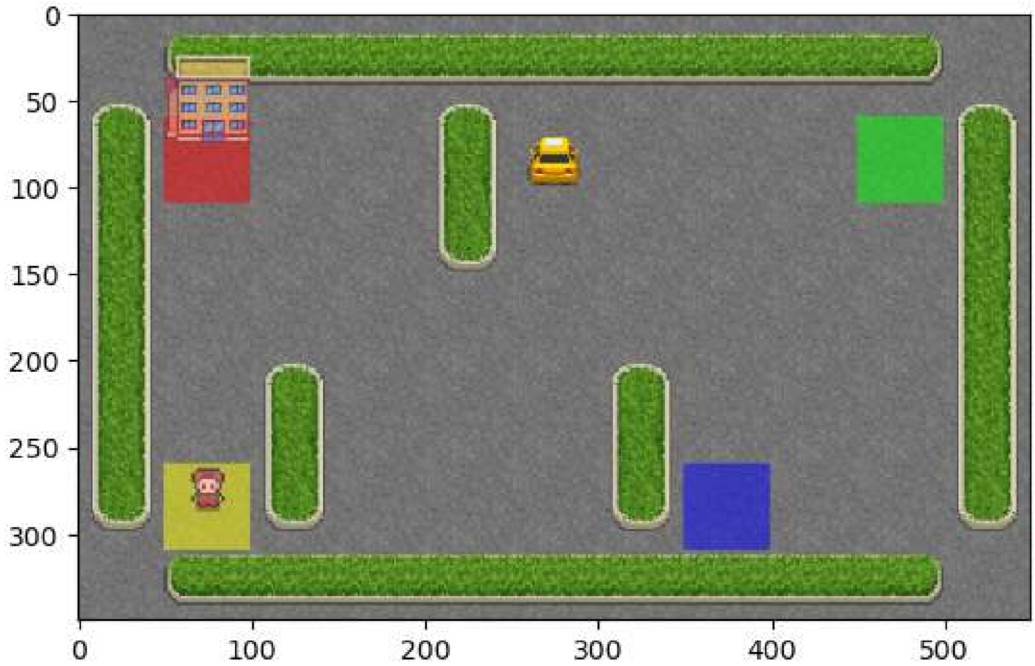
[7]:
```
<matplotlib.image.AxesImage at 0x10a73e450>
```



[8]:
```
next_state, reward, isTerminated, isTruncated, info=env.step(1)
print(next_state, reward, isTerminated, isTruncated, info)
image=env.render()
plt.imshow(image)
```

```
48 -1 False False {'prob': 1.0, 'action_mask': array([1, 0, 1, 0, 0, 0],
dtype=int8)}
```

[8]: <matplotlib.image.AxesImage at 0x146594210>



[9]:
```
import imageio
import numpy as np
images=[]

env.step(1)
images.append(env.render())

env.step(1)
images.append(env.render())

env.step(1)
images.append(env.render())

env.step(2)
images.append(env.render())

env.step(2)
images.append(env.render())
imageio.mimsave('./render.mp4', [np.array(img) for i, img in␣
↪enumerate(images)], fps=1)
```

```
IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by
macro_block_size=16, resizing from (550, 350) to (560, 352) to ensure video
compatibility with most codecs and players. To prevent resizing, make your input
image divisible by the macro_block_size or set the macro_block_size to 1
```

```
(risking incompatibility).
[swscaler @ 0x7fb6cbf8e000] Warning: data is not aligned! This can lead to a
speed loss
```

[10]:
```
from base64 import b64encode
from IPython.display import HTML
from IPython import display

# Show video
compressed_path = 'render.mp4'
mp4 = open(compressed_path,'rb').read()
data_url = "data:video/mp4;base64," + b64encode(mp4).decode()


HTML("""<video width=400 controls>      <source src="%s" type="video/mp4"></
 ↪video>""" % data_url)
```

[10]:
```
<IPython.core.display.HTML object>
```

[ ]:
```

```

xg0nx69i4

December 17, 2023

[ ]:
```
!pip install -r https://raw.githubusercontent.com/malkiAbdelhamid/
↪Advanced-Deep-Learning-2023-2024-esisba/master/lab1_QLearning/
↪requirements_lab1.txt
```

Before you solve a Reinforcement Learning problem you need to define what are

- the environment
- the states
- the actions
- the rewards

We are using the `FrozenLake-v1` environment from OpenAI's gym: https://www.gymlibrary.dev/environments/toy_text/frozen_lake/

`FrozenLake-v1` is an easy environment because the action space is small, and the state space is large but finite.

Environments with a finite number of actions and states are called tabular

0.0.1 Import the Gymnasium Library

[ ]:
```
import gymnasium as gym
from gymnasium.envs.toy_text.frozen_lake import  generate_random_map
```

0.0.2 Create a FrozenLake-v1 environment with gym.make()

- default map=4x4
- In order to display the environment's current state you need to add the parameter==> render_mode="rgb_array"

[ ]:
```
env=gym.make("FrozenLake-v1", is_slippery=False, render_mode="rgb_array")
#env=gym.make("FrozenLake-v1",desc=generate_random_map(size=8), map_name="8x8",␣
↪is_slippery=False, render_mode="rgb_array")
```

0.0.3 We reset the environment to its initial state with state = env.reset()

[ ]:
```
state=env.reset()
```

State space

- The state is a value representing the agent's current position as current_row * nrows + current_col (where both the row and col start at 0).
- For example, the goal position in the 8x8 map can be calculated as follows: 7 * 8 + 7 = 63. The number of possible observations is dependent on the size of the map. For example, the 8x8 map has 64 possible states.

[ ]:
```
print("State Space {}".format(env.observation_space.n))
print(env.observation_space.sample())
```

Action space :

The agent takes a 1-element vector for actions. The action space is (dir), where dir decides direction to move in which can be:

- 0: LEFT
- 1: DOWN
- 2: RIGHT
- 3: UP

[ ]:
```
print("Action Space {}".format(env.action_space.n))
print(env.action_space.sample())
```

env.render(): display the environment's current state

[ ]:
```
import matplotlib.pyplot as plt

image=env.render()
plt.imshow(image)
```

env.step(n_action)–> next state, reward, terminated,truncated, info

Updates an environment with actions returning:

- the next agent state,
- the reward for taking that actions,
- if the environment has terminated or truncated due to the latest action
- and information from the environment about the step

[ ]:
```
#apply the right action,
next_state, reward, isTerminated, isTruncated, _=env.step(2)
print(next_state, reward, isTerminated, isTruncated)
```

```
plt.imshow(env.render())
```

First, reset the environment, then define the trajectory which is a set of necessary actions required to achieve the goal

Finally, record the different steps through a video

[ ]:
```
import imageio
import numpy as np

env.reset()
images=[]
images.append(env.render())

#right, right, down, down, down, right

env.step(2)
images.append(env.render())

env.step(2)
images.append(env.render())

env.step(1)
images.append(env.render())

env.step(1)
images.append(env.render())

env.step(1)
images.append(env.render())

env.step(2)
images.append(env.render())

imageio.mimsave('./render.mp4', [np.array(img) for i, img in␣
 ↪enumerate(images)], fps=1)
```

[ ]:
```
from base64 import b64encode
from IPython.display import HTML

# Show video
compressed_path = 'replay.mp4'
mp4 = open(compressed_path,'rb').read()
data_url = "data:video/mp4;base64," + b64encode(mp4).decode()

HTML("""<video width=400 controls>      <source src="%s" type="video/mp4"></
 ↪video>""" % data_url)
```

0.1 vectorized environment gym

We create a vectorized environment (a method for stacking multiple independent environments into a single environment) of 16 environments, this way, we'll have more diverse experiences during the training. https://gymnasium.farama.org/api/vector/

[ ]:
```
envs=gym.vector.make('FrozenLake-v1',  num_envs=4)
env2=gym.make('FrozenLake-v1')
```

[ ]:
```
envs.reset(seed=42)
```

[ ]:
```
env.reset()
```