UNIVERSITY OF TEHRAN

COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NEURAL NETWORK & DEEP LEARNING

EXTRA ASSIGNMENT

MOHAMMAD HEYDARI

810197494

UNDER SUPERVISION OF:

DR. AHMAD KALHOR

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF TEHRAN

May. 2022

1 CONTENTS

2 QUESTION #1: LUNAR LANDER (DEEP Q-LEARNING)

In this part we intend to implement Deep Q-Learning algorithm in case of training an agent to solve Lunar-Lander problem.

Further, we are going to report the results of the sections one by one:

2.1 LUNAR LANDER ENVIRONMENT FEATURES

First of all, we should get familiar with the Lunar-Lander problem so we are going to have a brief look on its features:

Goal:

The goal of lunar lander is to land a small spacecraft between two flags.

Actions:

There are **four actions** :

1) fire your main engine
2) your left engine
3) your right engine
4) do nothing.

States:

Mixed Observation Space: Each observation contains 8 values:

(Continuous): X distance from target site

(Continuous): Y distance from target site

(Continuous): X velocity

(Continuous): Y velocity

(Continuous): Angle of ship

(Continuous): Angular velocity of Spacecraft

(Binary): Left leg is grounded

(Binary): Right leg is grounded

Rewards:

If the spacecraft hits the ground at the wrong angle or too fast it crashes and you get a large penalty. You also get a small penalty for firing your main engine. You get rewards for getting closer to the desired end position, and for touching the ground.

Solving-Condition:

The environment is considered solved if the agent manages to get a score of 200 or more on average in the last 100 episodes.

Some other features about environment:

Fully Observable: All necessary state information is known observed at every frame.

Single Agent: There is no competition or cooperation.

Deterministic: There is no stochasticity in the effects of actions or the rewards obtained.

Episodic: The reward is dependent only on the current state and action.

Static: There is no penalty or state change during action deliberation.

Finite Horizon: The episode terminates after a successful land, crash, or 1000 steps.

2.2 DQN ALGORITHM

In this part I have implemented DQN algorithm to investigate the performance of lunar-lander agent().

First of all, we must find the **suitable hyper-parameters** for training the model, just after few runs we found that we should use below parameters to scape from getting into the local-minimum.

n-episodes = 250

eps = 1.0

eps-decay-rate = 0.99

eps-end = 0.01

gamma = 0.99

Learning-rate =5e-4

Capacity = 25000

please note that one of the most important parameter in the task of training is about **choosing suitable value** for the **batch-size** and further we are going to choose between **three values** which abbreviated below:

1) batch-size = 32

2) batch-size = 64

3) batch-size = 128

We train the model for all of above batch-size values and then **using the regret metrics and also convergence rate metrics** determine whether we are dealing with a suitable-one or not!

Further, you can find the results of three separated cases:

Reporting the result:

Case one : **batch-size=32**

```
Episode 25      Average Reward: -178.36 Epsilon: 0.78
Episode 50      Average Reward: -144.82 Epsilon: 0.61
Episode 75      Average Reward: -78.40  Epsilon: 0.478
Episode 100     Average Reward: -82.20  Epsilon: 0.37
Episode 125     Average Reward: -7.36   Epsilon: 0.289
Episode 150     Average Reward: 52.20   Epsilon: 0.22
Episode 175     Average Reward: 155.31  Epsilon: 0.17
Episode 200     Average Reward: 134.54  Epsilon: 0.13
Episode 217     Average Reward: 204.45  Epsilon: 0.11
Environment solved in 217 episodes!     Average Score: 204.45
```

Figure1. representation of training process

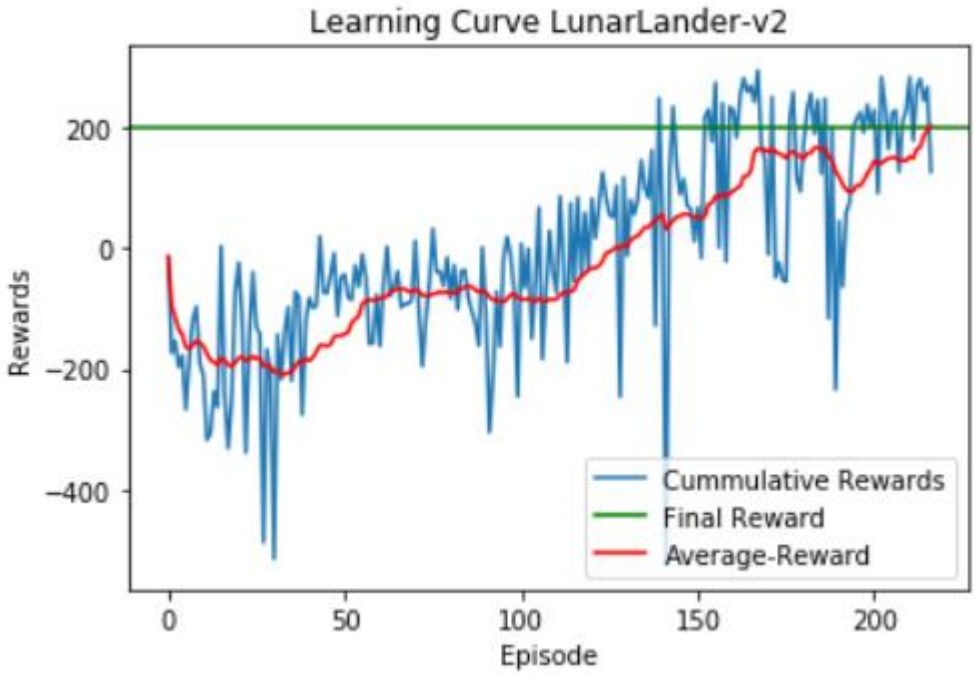

Figure2. learning curve LunarLander-v2

Reporting the result:

Case two : **batch-size=64**

```
Episode 25      Average Reward: -180.38 Epsilon: 0.78
Episode 50      Average Reward: -127.91 Epsilon: 0.61
Episode 75      Average Reward: -62.30  Epsilon: 0.475
Episode 100     Average Reward: -105.87 Epsilon: 0.37
Episode 125     Average Reward: 11.22   Epsilon: 0.2816
Episode 150     Average Reward: 81.70   Epsilon: 0.22
Episode 175     Average Reward: 44.57   Epsilon: 0.17
Episode 200     Average Reward: 184.35  Epsilon: 0.13
Episode 217     Average Reward: 205.49  Epsilon: 0.11
Environment solved in 217 episodes!     Average Score: 205.49
```

Figure3. representation of training process

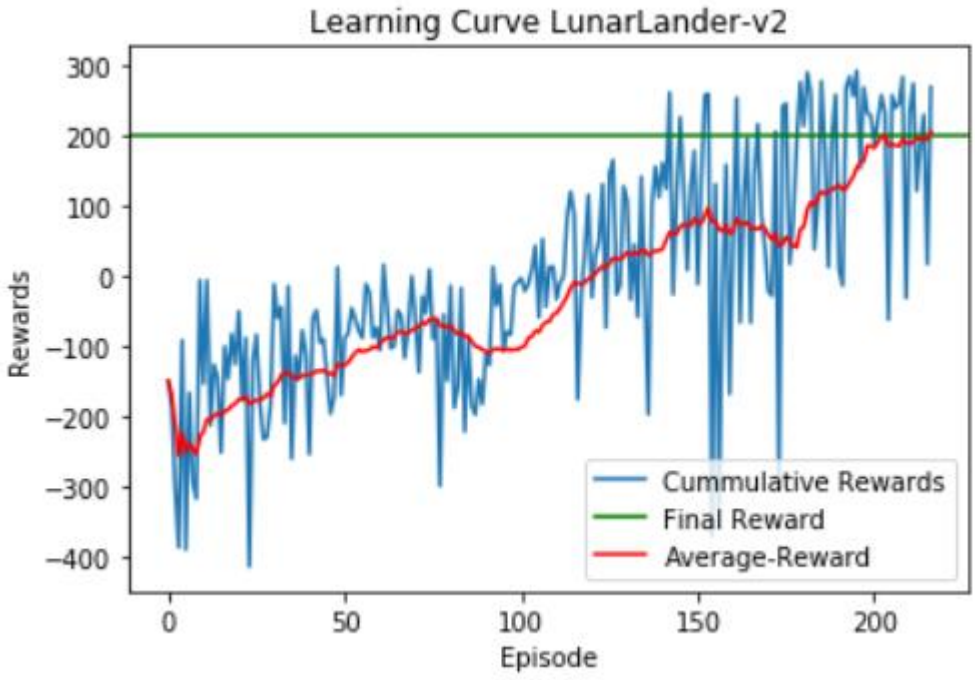

Figure4. learning curve LunarLander-v2

Reporting the result:

Case three : **batch-size=128**

```
Episode 25      Average Reward: -221.70 Epsilon: 0.78
Episode 50      Average Reward: -101.01 Epsilon: 0.61
Episode 75      Average Reward: -60.70  Epsilon: 0.478
Episode 100     Average Reward: -63.57  Epsilon: 0.37
Episode 125     Average Reward: 55.72   Epsilon: 0.282
Episode 150     Average Reward: 76.23   Epsilon: 0.22
Episode 175     Average Reward: 79.93   Epsilon: 0.17
Episode 200     Average Reward: 162.10  Epsilon: 0.13
Episode 208     Average Reward: 201.02  Epsilon: 0.12
Environment solved in 208 episodes!     Average Score: 201.02
```

Figure5. representation of training process

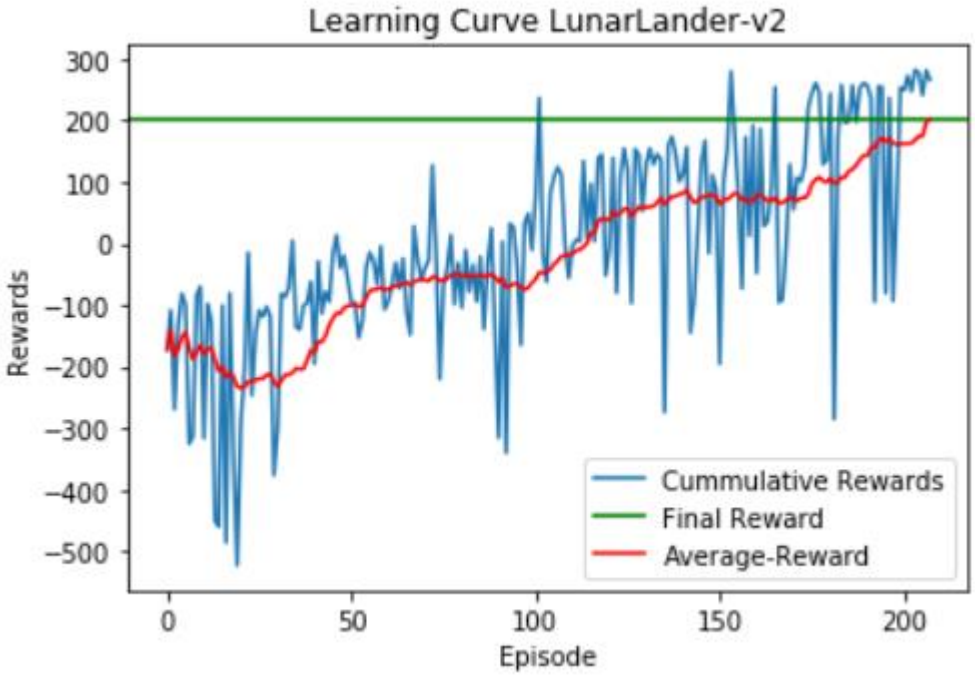

Figure6. learning curve LunarLander-v2

Comparing the results and choosing the best one:

in this section we are going to choose one of the three batch-size values according to the **regret metrics** and also **convergence rate** metrics.

1) Convergence-rate:

In this section we investigate that which-one reaches to the convergence in a smaller number of epochs.

As we can see **the batch-size=64** has a better convergence rate in comparison to others.

So, in case of convergence-rate **we choose batch-size =64 as of our optimum-value.**

2) Regret metrics:

First of all, let's get familiar with the regret metrics and then discuss about different batch-size values!

Regret:

the regret is expressed as the difference between the payoff (reward or return) of a possible action and the payoff of the action that has been actually taken.

So, we must investigate the area of between the horizontal line and the average-reward curve and the best-cases are those that have a smaller area because that means we are dealing with a better convergence rate.

So according to the above figures we should choose the **batch-size value =64** and it is same with the results that have concluded in the previous part.

Please note that **after choosing the optimum batch-size** value=64, I have used that in all of the next parts.

2.3 DQN VS DDQN IN CASE OF PERFORMANCE

In this section we are going to implement another approach which called DDQN algorithm and then compare the performance of model using both approaches.

As we found in the previous part**, the optimum value of batch-size** is equal to **64** and then according to the question description we choose the batch-size=64 for the both cases.

Further you can find the results of this section:

```
Episode 25      Average Reward: -178.60 Epsilon: 0.78
Episode 50      Average Reward: -87.46  Epsilon: 0.616
Episode 75      Average Reward: -74.75  Epsilon: 0.47
Episode 100     Average Reward: -43.69  Epsilon: 0.37
Episode 125     Average Reward: 46.12   Epsilon: 0.282
Episode 150     Average Reward: 49.82   Epsilon: 0.22
Episode 175     Average Reward: 32.01   Epsilon: 0.17
Episode 200     Average Reward: 115.85  Epsilon: 0.13
Episode 221     Average Reward: 207.75  Epsilon: 0.11
Environment solved in 221 episodes!     Average Score: 207.75
```

Figure7. representation of training process (DDQN)

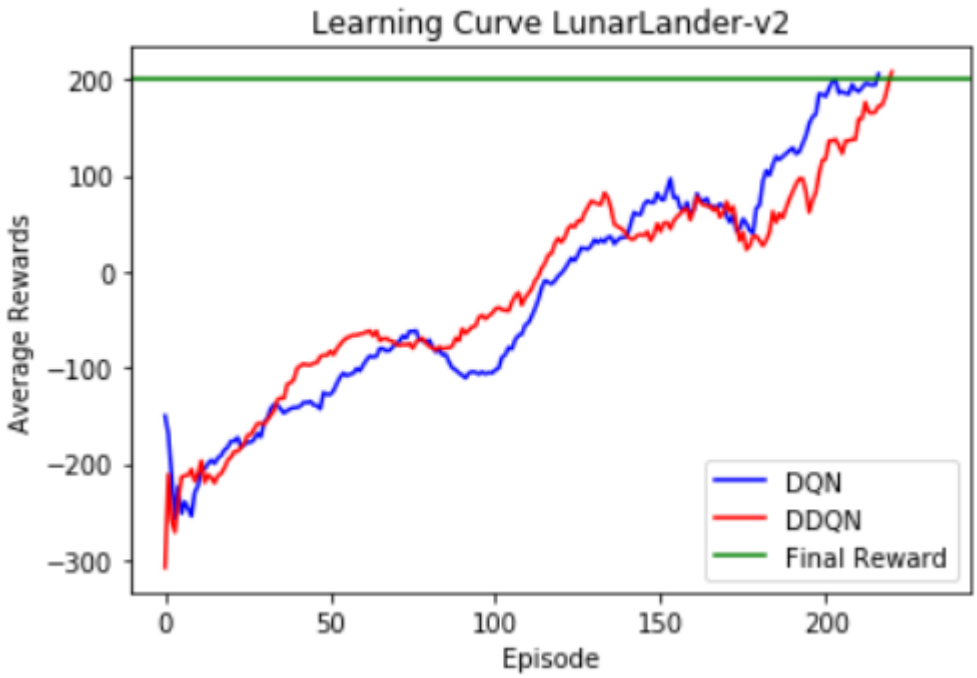

Figure8. learning curve LunarLander-v2 DDQN vs DQN(average rewards)

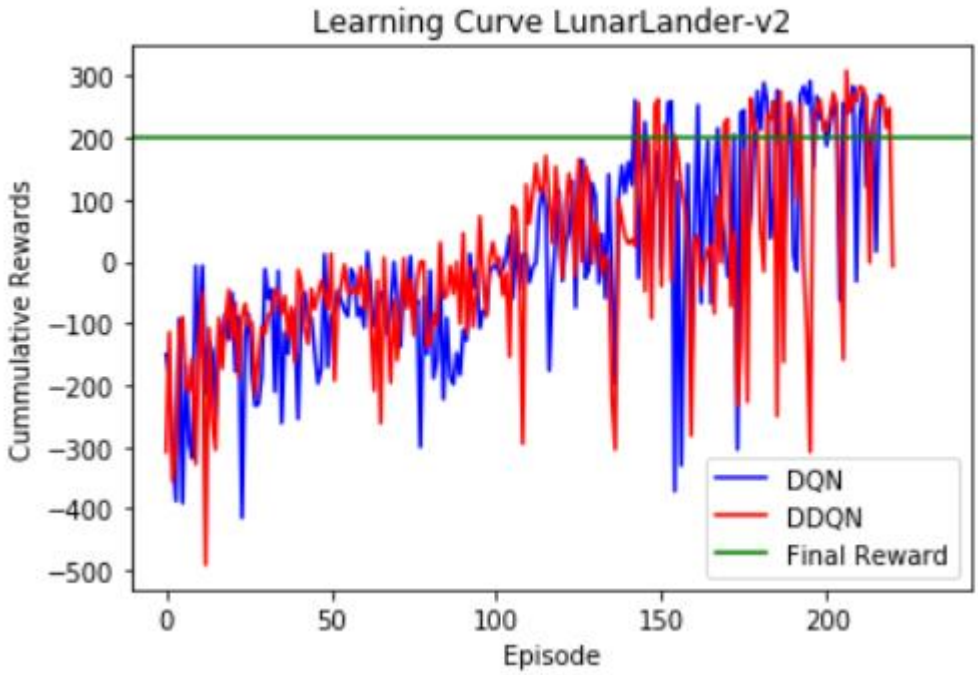

Figure9. learning curve LunarLander-v2 DDQN vs DQN(cumulative rewards)

Analyse of results:

As we can see **the target network** in this problem does not make **any significant improvement in case of performance and convergence rate**. and that is all about the type of our problem **which is very simple**!

In other word if we **consider regret metrics** and have a comparison between DQN and DDQN we found that the **difference between optimum-reward and average reward** is approximately equal and so we have no improvement in this case and as I mentioned before that is all about that the problem which is as much as simple that DDQN can cause any improvement.

3 REFERENCES

[1] https://towardsdatascience.com/ai-learning-to-land-a-rocket-reinforcement-learning-84d61f97d055

[2] https://towardsdatascience.com/introduction-to-regret-in-reinforcement-learning-f5b4a28953cd

[3] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.legend.html

[4] https://github.com/aio-libs/aiohttp/issues/2139

[5] https://wandb.ai/timssweeney/lunar-lander/reports/Lunar-Lander-Open-AI--VmlldzoyNzg5NjE

[6] https://shiva-verma.medium.com/solving-lunar-lander-openaigym-reinforcement-learning-785675066197