Solving the LunarLanderContinuous-v2

Eliyahu Shneider(ID. 302516463),Lior Shimon (ID. 341348498)

final project report for the RL course, BIU, 2021

1 Introduction

To master the tools learnt during the Reinforcement Learning course, as well as understanding the different hues, solving lunar lander was a very good way to get insights of the different mechanism RL is using. This is the report of our work on this Final project.

2 Solution

2.1 General approach

In order to solve the continuous action space environment, we decided to quantify the action into a finite number of states. At first, out of curiosity, we thought about quantifying the position of the spaceship in the environment and trying to solve the game with a model base approach. But since their are many states the transition table gets huge (even after strong quantification), we concluded that we have to move to the sample based approach. We planned to test different model : Deep-Q-Netword (dqn) , dueling deep Q Network (ddqn), SARSA , Policy Gradient and finally Actor Critic.

As DQN showed the best results, we mainly focused on it, trying to improve its results as best as possible.

2.2 Design

We worked on Pycharm IDE , working with pytorch DL framework. We built an agent, working on the gym environment , using a neural network to approximate its Q function and navigate through the environment.

since we started our project from scratch and this was our first Reinforcement Learning project, it took us roughly 60 hours to finally have a successfully working agent (average reward of 200 over a 100 episodes).

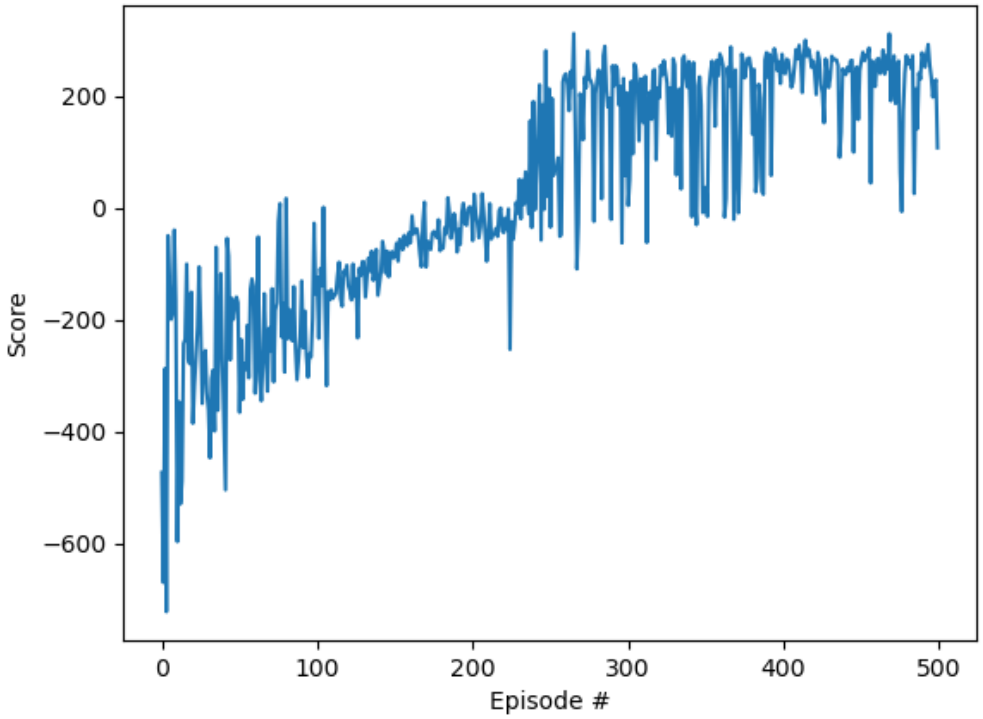
2.3 Results: outlook

in the deterministic Transition model, our agent solved the environment in 232 epochs using a DQN model. in the stochastic Transition model (Lunar-Lander

with uncertainty), our agent solved the environment in 398 epochs with the same model and the same hyper-parameters than the deterministic transition model.

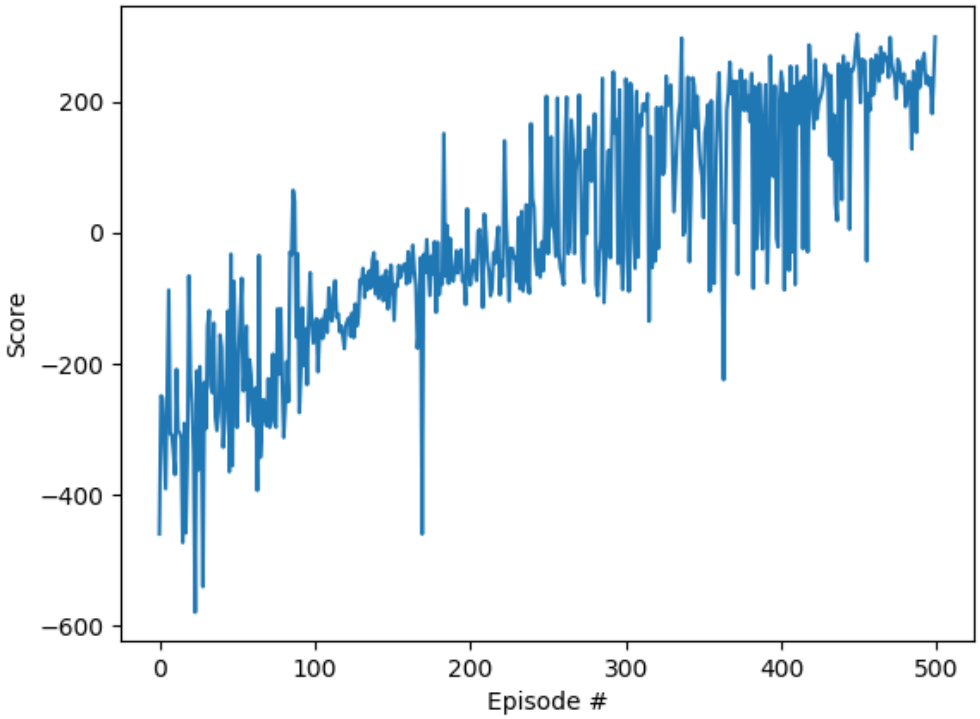
3 Experimental results

We solved (average reward of 200 over a 100 episodes.) in 232 epochs. The wining agent was DQN agent, and specifically Dueling DQN. The Policy Gradient and Actor Critic converged, but much much slower.



Our agent solved in 232 epochs with Dueling DQN model

Regarding the Lunar-Lander with uncertainty, we worked on our most successful model: the DDQN agent and its specific hyper-parameters. We achieve an average reward of 200 over a 100 episodes in 398 episodes.



With uncertainty, the agent solved in 398 epochs

4 Discussion

4.1 Policy gradient VS Deep Q Network

As we learned, Policy gradients methods (including Actor Critic) are good in cases where its hard to design good rewards system. In our case the reward system helped the agent to get better local minima's until it reached the required result. Suppose the reward system is +1000 if we achieve a good landing and -1 otherwise, the DQN may have bad results. A stochastic policy (not the case here) would get bad results as well with this reward system.

4.2 main challenge

we got stuck in the learning task, where the agent decided to stay in the air for getting as much reward as possible, leading to a 1000 steps in many episodes.

4.3 Playing with the rewards function

As our agent was stuck in the local minima of staying in the air, we tried to encourage it not to stay there. We tried penalties on high speed, bigger penalties on fuel usage, penalties on reaching the 100 episodes without landing, penalties on non vertical angles. In practise those didn't help, which make us believe the default rewards system was chosen wisely.

4.4 Playing with extending the states

In order to encourage the agent to try minimize episodes length We also tried to extend the state data to include more information. we tried adding a virtual fuel tank,in which the system would have a value in the state that decrease in every use of engine, and when it reach zero there is a big negative reward. We also tried to add the number of step in a single episode as a state parameter, giving a big penalty when finishing after more than 300 steps. Lastly, we tried to give a negative reward to the agent every time it reaches a number of step that is a multiple of 100, without any satisfying results. Since all of those extensions didn't helped, and since the agent converged without them at some point, we concluded the current state's structure of the Markov Decision process were sufficient .

4.5 Hyper Parameters

In every step in the process, we had to tune many parameters.

RL oriented parameters :

- Epsilon start, decay, stop
- Gamma
- steps until target update
- max steps in an episode
- memory buffer size and batch size

ML oriented parameters:

- optimizer (SGD, Adam, adagard)
- learning rate (due to the fact we use ML)
- network type (DQN, DDQM, Noisy DQN, PG etc)
- network architecture (layers number, layers sizes, dropout, activation funcs)

Game specific parameters:

- action spectrum

- reward system

This optimization process was very hard as each of those changes the results heavily. Furthermore, since this is an RL problem, the convergence is not obvious from the graphs.

5 Code

https://github.com/eliyash/RL-LunarLander-v2.git