ESI 6684: Decision Making with Deep Reinforcement Learning

Professors: Ankit Shah

Final Project: Luna Lander

Quoc H. Nguyen

Nov 26, 2023

Contents

1 Introduction

Lunar Lander v2 is an OpenAI Box2D environment simulating the optimization of rocket trajectory—a classic subject in Optimal Control. This environment includes a randomly generated terrain featuring a rocket and a landing pad. The lander is equipped with three engines: left, right, and bottom. The primary objective is to skillfully utilize these engines to land the rocket on the landing pad, positioned consistently at coordinates (0,0), while minimizing fuel consumption. Notably, landing outside the designated pad is permissible, and the agent benefits from infinite fuel, enabling the opportunity for the agent to learn how to navigate and successfully land on its initial attempt.

The state space consists of eight values:

- Position x
- Position y
- Horizontal velocity
- Vertical velocity
- Orientation in space (angle)
- Angular velocity
- Is left leg touching the ground?: 0/1 (Boolean)
- Is right leg touching the ground?: 0/1 (Boolean)

The action space is Box(-1, +1, (2,), dtype=np.float32):

- The first coordinate determines the throttle of the main engine.
- The second coordinate specifies the throttle of the lateral boosters.

The reward is generated by the environment and is a sum of the following components:

- Closer (higher reward)/further (lower reward) the lander is to the landing pad
- Slower (higher reward)/faster (lower reward) the lander is moving
- Less (higher reward)/more (lower reward) tilt of the lander
- If lander crashes or comes to rest, it gets -100 or +100
- Each leg with ground contact gets +10
- Firing the main engine is -0.3 per frame
- Firing the side engine is -0.03 per frame

2 Deep Deterministic Policy Gradient (DDPG)

The Deep Deterministic Policy Gradient (DDPG) algorithm is a combination of elements from both the Deep Q-Network (DQN) and Actor-Critic algorithms. Since DQN (***I have implemented DQN, but it only works well with discrete actions, and could not get convergent when I tried to convert the continuous action spaces to discrete. Please see Q_DQN_LunaLander.ipynb file***) is not suitable for environments with continuous action spaces due to its reliance on a stochastic policy, DDPG was introduced, featuring a hybrid deterministic policy. In contrast to a stochastic policy that generates a probability distribution for action sampling, DDPG employs a deterministic policy.

To ensure sufficient exploration with the deterministic policy, which outputs actions based on states, random function noise is added to the obtained actions. The DDPG paper specifically mentions the use of an Ornstein-Uhlenbeck process for generating this noise. DDPG involves a total of four neural networks. DQN utilizes two networks: a moving neural

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 1: DDPG algorithm

network (updated at each step) and a target neural network (slowly tracking the moving

network). In Actor-Critic algorithms, there are also two networks: an Actor (policy network generating actions based on the given state) and a Critic (value function network estimating the value of a given state). Consequently, DDPG employs four neural networks—both moving and target networks for both the Actor and Critic components. The target networks follow the moving networks through polyak averaging, ensuring a smoother learning process.

3 Results

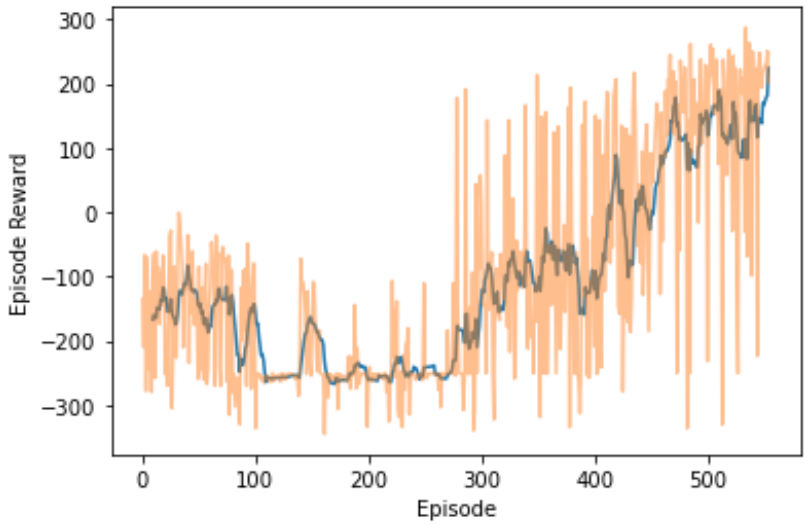

Figure 2: DDPG training

Observations (Please see the output in detail in source code files):

- During multiple runs, mean test return is over 200, therefore we can conclude that goal is reached! The average training time to get the rewards of 200 is around 30 minutes using RTX 3060.
- Fastest run reached the goal after 71,435 steps (194 episodes).
- Highest reward in a single episode achieved is 315.
- Please see my hyper-parameter in the code files. It took me awhile to find out the best set of hyper-parameters with trial-errors and information in the original DDPG paper.
- The standard DDPG implementation involves using Ornstein Ulhenbeck (OU) noise for exploration. I think we can improve the performance with using Gaussian noise for exploration.