



Lunar Landing with Reinforcement Learning

07.15.2019

Sergio Robledo

Domain Background

Landing any sort of aircraft has always been a task that requires some sort of skilled pilot. For example, Apollo 11 needed the pilots to know how to control the spacecraft in order for them not to crash onto the face of the moon. More recently, SpaceX has decreased the cost of space rocket construction through arming their rockets with the capability of landing themselves for future use which eliminates the need to build a new space rocket for every mission. I will be attempting to train a deep reinforcement learning algorithm to land a lunar spacecraft found in [OpenAI's environments](#).

After spending countless hours developing a deep deterministic policy gradient, or DDPG, algorithm to take a quadcopter from the ground to a desired height, I believe a variation of that DDPG algorithm could also be used to train the landing lunar module to land at a certain position desired by the user. Making reinforcement learning technology work at a smaller scale without consequences such as the lunar landing is an essential step into possible future implementation of reinforcement learning in the real world.

Problem Statement

The lunar landing problem entails a spacecraft freefalling from a certain distance above the ground, and the end goal is for the spacecraft to maneuver, with the use of a left, right and main engine, to the target position of (0,0) representing the ground area in between the target flags. The performance of the algorithm used is based off of the reward the lunar landing environment gives, and the problem is considered solved when the agent can repeatedly land the spacecraft with a score of 200 or more.

Datasets and Inputs

The only data I will be using comes from the OpenAI Lunar Lander environment. This environment provides the physics mechanism that will be affected by the actions of the learning algorithm. The initial state will be provided by the environment which will serve as the starting input of the learning agent. Once the state is given, the agent will predict the best action for the spacecraft to make, whether it be turning on the right engine, left engine, main engine or neither, and this action will be passed back to the environment which will output the next state of the spacecraft. This loop will be repeated until the agent learns what actions will maximize its reward.

Solution Statement

In order for the agent to learn to land, the agent will use a prioritized experience replay buffer, action noise created through the Ornstein-Uhlenbeck process and will be comprised of a given number of neural networks. The proposed neural network setup will include an actor network that makes decisions, along with a critic network that evaluates those decisions. Each actor and critic network will have an identical neural network that will serve as the targets of the agent who will be updated slowly in order to improve the stability of the learning agent. The problem will be considered solved, once the agent is able to consistently achieve a score of 200 or more in the landing of the spacecraft.

Benchmark Model


There is a [GitHub repository](#) that showcases the highest score achieved by the people who have attempted this problem. Many of them use variations of reinforcement learning algorithms ranging from DQNs to PPOs. There are many of them who have solved the lunar lander environment with a score of 200+, and the only difference is the number of episodes it took their learning agent to achieve such a score. Once I create and optimize my lunar lander algorithm, I can compare my results with the results of the best performing algorithms seen in the leaderboards.

Evaluation Metrics

The main performance metric will be the reward obtained by the RL agent in a given episode of the environment. For example, in the GitHub repository linked above, it can be seen that the main two metrics displayed when showcasing the performance of the algorithms created by others are the reward and number of episodes taken to reach that reward. In order for my algorithm to be considered successful, it would have to get a higher reward on its last episode than it did on the starting one.

Project Design

In order to succeed in this project, I will have to consider various reinforcement learning architectures and decide whether or not to discretize the action space since this would lead to a divergence as to which algorithms would be able to solve such a problem and which could not. In order to use samples more efficiently, I will implement a prioritized experience replay buffer which gives a proportional importance to each sample based on



how much the reward changed in that experience; the higher the reward change the higher the priority given to that sample. Also, in order to encourage exploration, I will add noise to the state inputs and/or the action outputs through the Ornstein-Uhlenbeck process. I will be using the Keras library to build and train the neural networks.