# Machine Learning Engineer Nanodegree

## Capstone Proposal

Jean Ricardo Rusczak
February 21st, 2019

## Proposal

### Domain Background

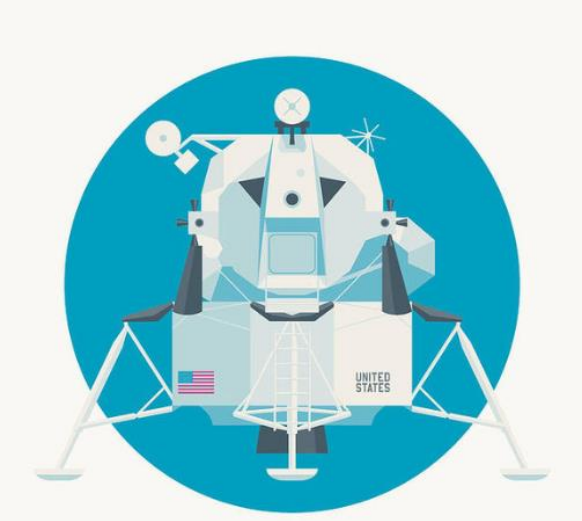


*Figure 1 - Apollo 11 Lunar Lander Artwork*

*Source: https://fineartamerica.com/featured/apollo-lunar-module-lander-minimal-text-cyan-ivan-krpan.html?product=art-print*

The ability to have a controlled descent in any surface has been for a long time object of study in the aerospace industry. For manned missions it is possible to rely the skills of an experienced pilot, but for remote controlled missions the vehicle has to be able to land by itself. This is mostly because the communication delay between Earth and the spacecraft, which can be from several minutes to hours, depending where the spacecraft will land.

In the recent years, the company SpaceX proved it is possible to improve the rocket reusability by performing a soft landing by using advanced adaptive algorithms [1]. The rocket family Falcon 9 has been able to reuse its boosters due to 33 landings of 40 attempts [2].

Other important milestones are the spacecraft Philae [3] that managed to land on comet 67P/Churyumov–Gerasimenko on 2014, and the rovers and probes sent to

Mars. A good example of powered descent was the InSight lander [4][5], that had a smooth and precise landing on Mars on Nov, 2018.

## Problem Statement

For this study it will be used a simplified physical environment to simulate the landing of a spacecraft on the Lunar surface using the OpenAI environment called Gym [6].

Lunar Lander Requirements:

- The landing pad shall have fixed coordinates (0,0).
- The lander can start in random positions.
- The lander shall be rewarded for going from the top of the screen to landing pad with zero speed, and the reward varies from 100 to 140 points.
- The lander shall control its X,Y positions by activating the left and right engines in order to land precisely on the landing pad.
- The episode shall finish if the lander crashes or comes to rest, receiving additional -100 or +100 respectively.
- For each leg contact with ground, the lander shall receive +10 points.
- The lander shall be penalized in 0.3 points each time (step) the main engine is ON.
- Landing outside of the landing pad is allowed, but it shall not receive the maximum reward.
- Fuel can be infinite.

## Datasets and Inputs

*Dataset*

There is no previous dataset and the agent will learn based on interactions with the environment. For helping improving convergence, an experience replay buffer can be used to improve the agent decision-making process.

*Inputs*

- The action space is continuous from -1.0 to +1.0 defined in two real values.
- The first one defines the behavior of the main engine, -1 to 0 and 0 to +1. Being 0 the OFF state for this engine.
- Engine cannot work with less than 50% of power.
- The second value is defined by -1.0 to -0.5 to fire the left engine, +0.5 to +1.0 to fire the right engine, and -0.5 to +0.5 OFF.

## Solution Statement

The classic reinforcement learning problem is defined by an agent applying actions to an environment and observing the new system state and also how he can be rewarded for that specific action.

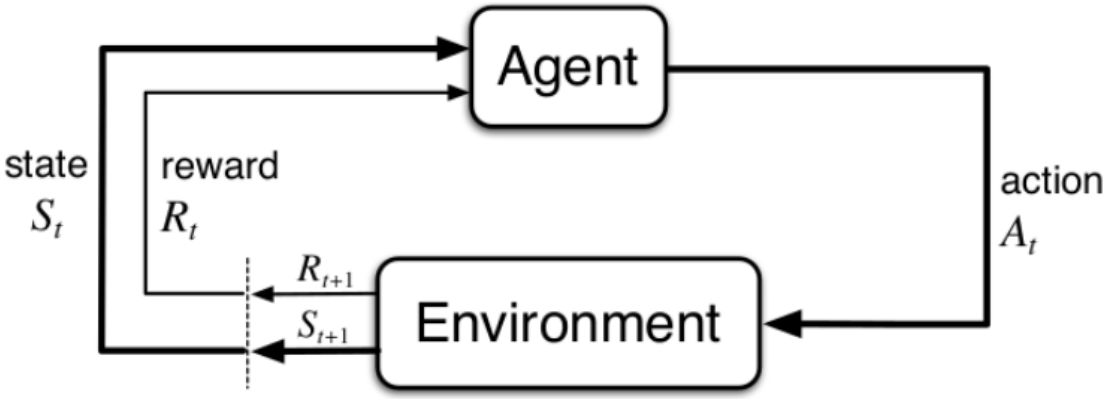


*Figure 2 – Reinforcement Learning Problem*

Depending of the type of observation and action spaces required for that particular environment, a different type of algorithm should be used:

| Algorithm | Model | Policy | Action Space | Observation Space | Operator |
|---|---|---|---|---|---|
| Q-Learning | Model-Free | Off-policy | Discrete | Discrete | Q-value |
| SARSA | Model-Free | On-policy | Discrete | Discrete | Q-value |
| DQN | Model-Free | Off-policy | Discrete | Continous | Q-value |
| DDPG | Model-Free | Off-policy | Continous | Continous | Q-value |
| TRPO | Model-Free | Off-policy | Continous | Continous | Advantage |
| PPO | Model-Free | Off-policy | Continous | Continous | Advantage |

*Figure 3 - Main Reinforcement Learning Algorithms*

*Source: https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-part-ii-trpo-ppo-87f2c5919bb9*

In this case, as both action and state space are continuous, there are three types of algorithm that can be used:

- Deep Deterministic Policy Gradients (DDPG)
- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)

In this project at least DDPG will be investigated and tuned to perform the landing task based on the study proposed by [6].

The basic concept of DDPG architecture is shown in the Figure 4.

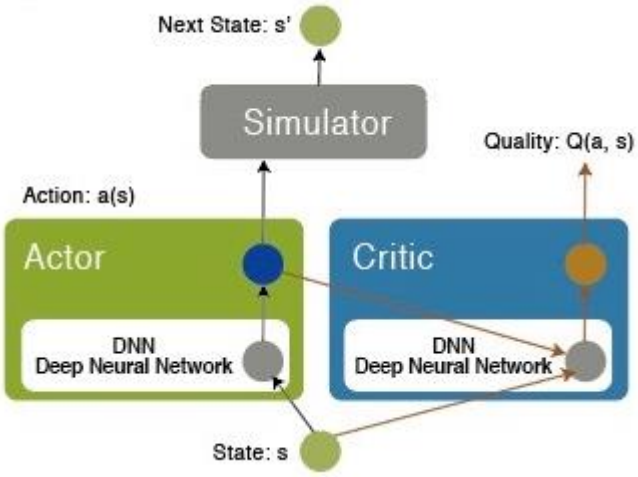


*Figure 4 - DDPG Architecture*

*Source: https://gridpredict.jp/our_service_en/making-ai-experience-many-trials-to-achieve-high-level-behavior/*

## Benchmark Model

For the specific environment there are only a few benchmarks available [8], which gives as the best result a score greater than 200 for 100 trials. The author uses the Proximal Policy Optimization algorithm to achieve that.

## Evaluation Metrics

**Success/Failure per episode**: by the end of any episode the lander will succeed or fail, these numbers should be stored for algorithm comparison.

**Time to complete task**: faster the module lands successfully, less fuel is required and more efficient the algorithm.

**Number of actions taken per episode**: this metric will verify how stable the controller is. If many actions were required, might mean that the controller is a near-instability region.

**Cumulative reward per episode**: as defined in the section Problem Statement, several factors define a successful landing, so higher the cumulative reward, closer to the ideal landing the module is.

## Project Design

The basic design for this project will be the one described in Figure 2, where the environment is the lunar lander module from OpenAi-Gym and the agent is the DDPG algorithm.

**Algorithm 1** DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ 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 5 - DDPG Algorithm [6]*

## References


[1] Açıkmeşe, Behçet & Casoliva, Jordi & Carson, John & Blackmore, Lars. (2012). **G-FOLD: A Real-Time Implementable Fuel Optimal Large Divert Guidance Algorithm for Planetary Pinpoint Landing.**

[2] https://spacexnow.com/stats.php

[3] https://en.wikipedia.org/wiki/Philae_(spacecraft)

[4] https://mars.nasa.gov/insight/timeline/landing/entry-descent-landing/

[5] https://mars.nasa.gov/insight/timeline/landing/summary/

[6] https://gym.openai.com/envs/LunarLanderContinuous-v2/

[7] Lillicrap, Timothy P., et al. (2015). **Continuous Control with Deep Reinforcement Learning.**

[8] https://github.com/openai/gym/wiki/Leaderboard