Learning to do soft landing on the Moon

By, Shashank Raghuvanshi

Introduction

Reinforcement Learning is a field of machine learning in which an agent (or multiple agents) interact within an uncertain environment (and/or with each other), to maximize some notion of a cumulative reward [1].The Lunar Lander gym environment has been designed to test Reinforcement Learning algorithms for solving rocket trajectory optimization problems, which is a classical optimal control problem [2]. The project aims to compare different Reinforcement Learning (RL) Algorithms like SARSA, DQN and PPO on the given problem, and find which algorithm is efficient to solve the lunar lander environment. It's intended to be an exercise to gain practical insights into different Reinforcement Learning Algorithms and Open AI Gym environments.

Problem Statement

Lunar Lander environment simulates a situation in which a lander has to land softly in a low gravity environment at a designated landing pad, for this the environment has a well defined physics engine implemented. The main goal is to make the agent do a soft landing on a designated landing pad, using as little fuel as possible. This is a classical environment to test different RL Algorithms. The environment has continuous state space with discrete action space for the agent.

Motivation

As someone who just got introduced to the concepts of Reinforcement Learning, this environment provides a suitable ground to test out the popular RL algorithms. The environment strikes a right balance between keeping the simulation real, and at the same time not being very complex.

Environment Details

State Space

The state space for the observation in the environment has 8 parameters, of which 6 are continuous values and 2 are boolean values. If we assume s to be a 8 tuple observation value (indexed from 0) then:

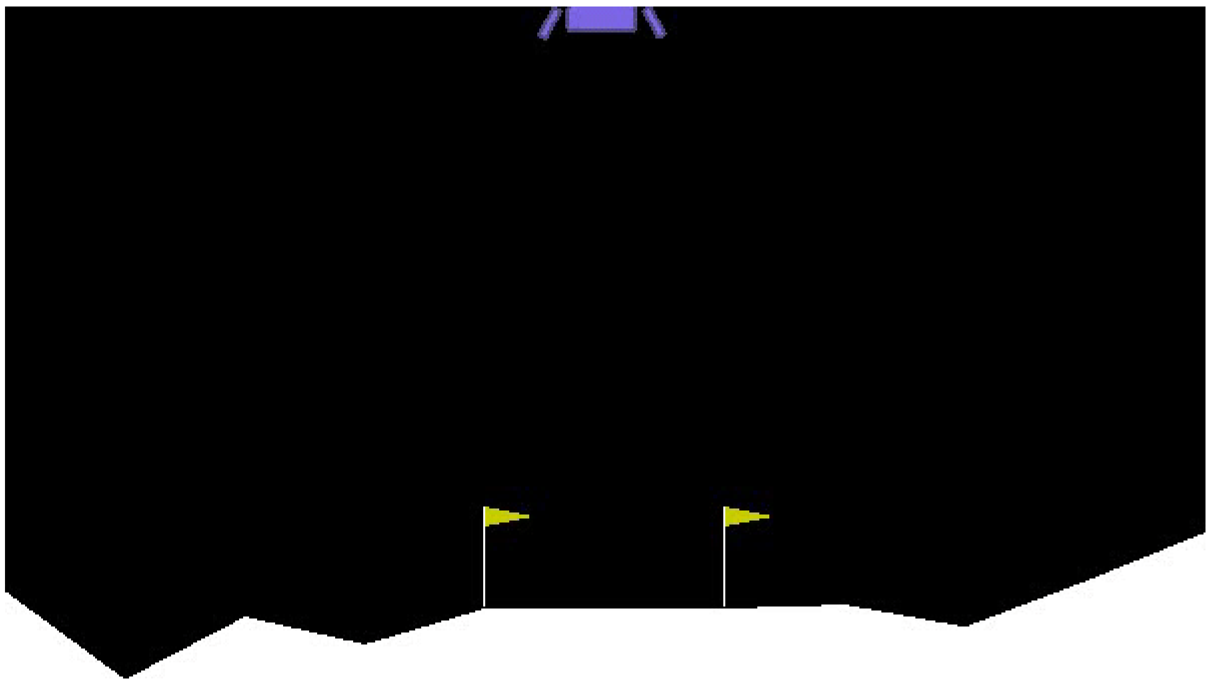
s[0] is the horizontal coordinate (continuous)
s[1] is the vertical coordinate (continuous)
s[2] is the horizontal speed (continuous)
s[3] is the vertical speed (continuous)
s[4] is the angle (continuous)
s[5] is the angular speed (continuous)
s[6] 1 if first leg has contact, else 0 (boolean)
s[7] 1 if second leg has contact, else 0 (boolean)

Action Space

Action space is discrete, and consists of 4 possible actions, that an agent can take which are **Do nothing, Fire Left Engine, Fire Right Engine or Fire Main Engine**. It is relevant to point out here that in order to make the task challenging for the agent, the left and right engines also add torque to the agent when used, which makes it a little more challenging to gain stability.

Reward Criteria

A proper reward criteria is the key to make the agent learn the required behaviour. The agent needs to maintain a stable posture and reach the landing pad as quickly and efficiently as possible. When the agent crash lands then a -100 point penalty is imposed. Similarly on safe landing the agent gains +100 points. For each leg of the agent if there is a contact with the ground then it receives +10 points. Firing the main engine imposes a slight penalty of -0.3 points for each frame of the episode, to encourage the agent to use less fuel.



Lunar Lander Environment with the agent choosing a random action.

Methodology

I tested 3 algorithms to solve the Lunar Lander environment, which are SARSA, DQN and PPO. For SARSA the discretization of observation space was required, however in PPO and DQN observation space was not discretized.

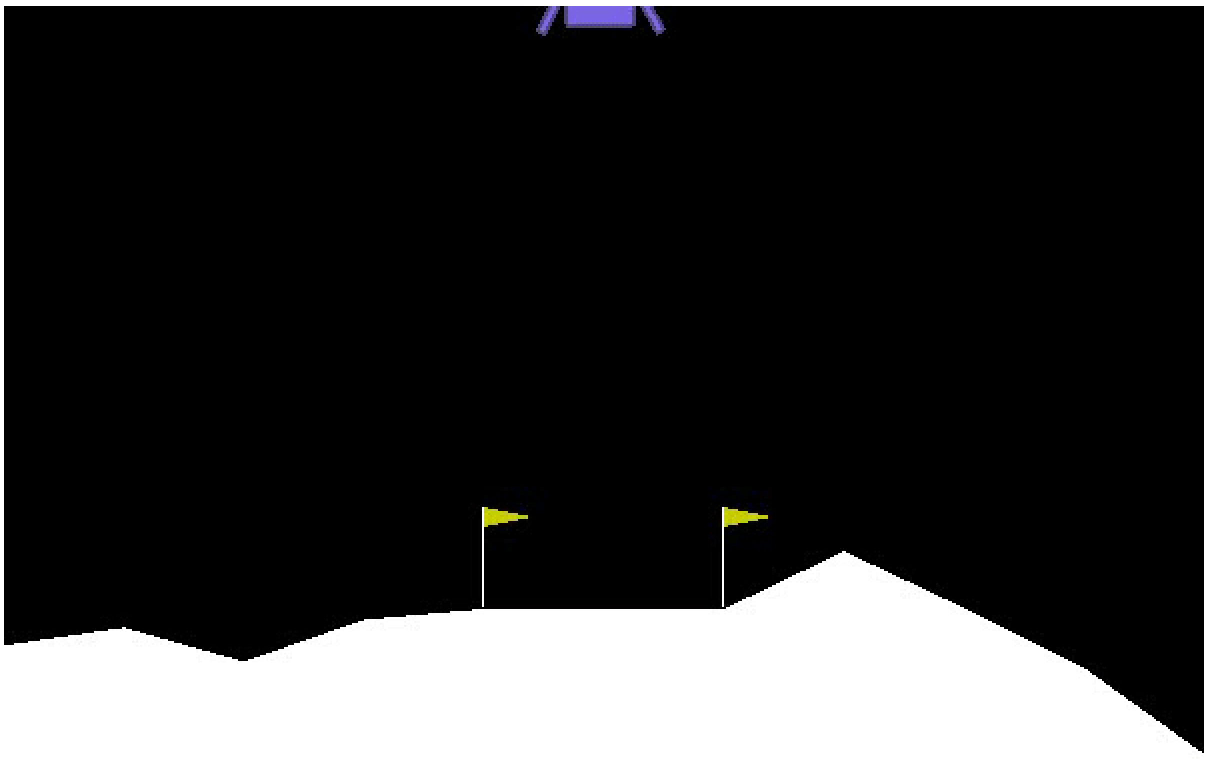
SARSA

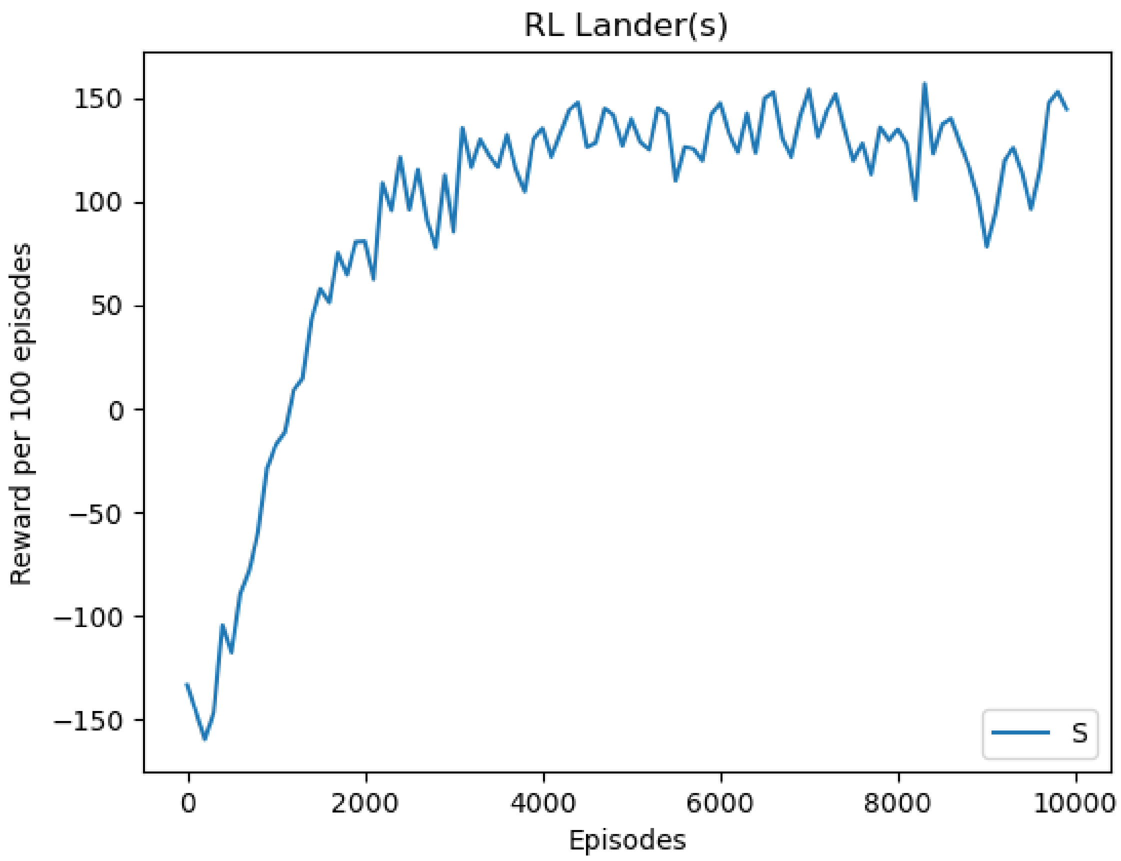
For SARSA the discretization of observation space as discussed in the paper[1] was used. In order to confirm the suggested range of values, the agent was tested with different actions to take it to an extreme boundary of each value in practice, before the episode ends. The observations confirmed that the suggested ranges should work. The horizontal axis values were discretized between -2 and +2 with a total number of 80 discrete values. Similarly the other continuous values were discretized to 40 discrete values between the same range. This optimization turned out very effective for the SARSA algorithm. SARSA is an on policy algorithm which requires a lot of exploration of the environment. The Q function update value is given by the following equation

$$Q(S,A) = Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$$

Here α is the learning rate, γ is the hyperparameter to discount future reward. For this project the learning rate(α) was 0.1 and gamma (γ) was 0.99. The algorithm was trained with 10k episodes. In order to balance between exploration and exploitation of the learned q-function for making a decision, another hyperparameter epsilon is used, with a decay rate of 0.99 . If epsilon is equal to 1 then agent always chooses an action at random, if the epsilon is 0 then agent always chooses an action based on the learned q-function (exploitation). For training the epsilon is initially kept large, and then slowly decayed until it reaches a minimum value of 0.01.

SARSA Results:

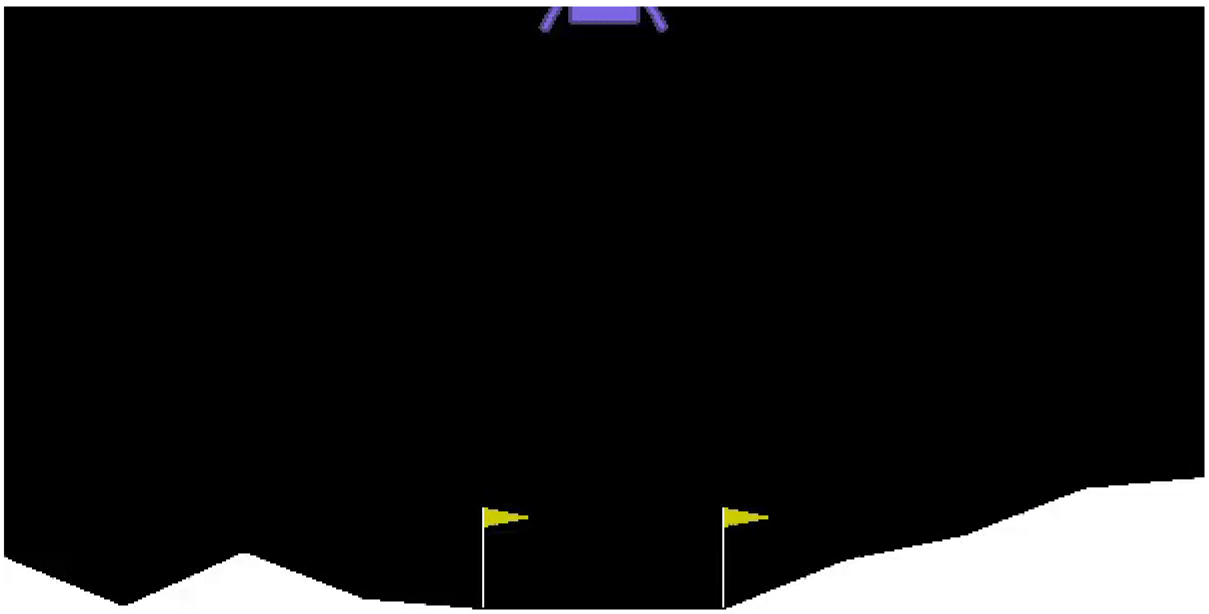


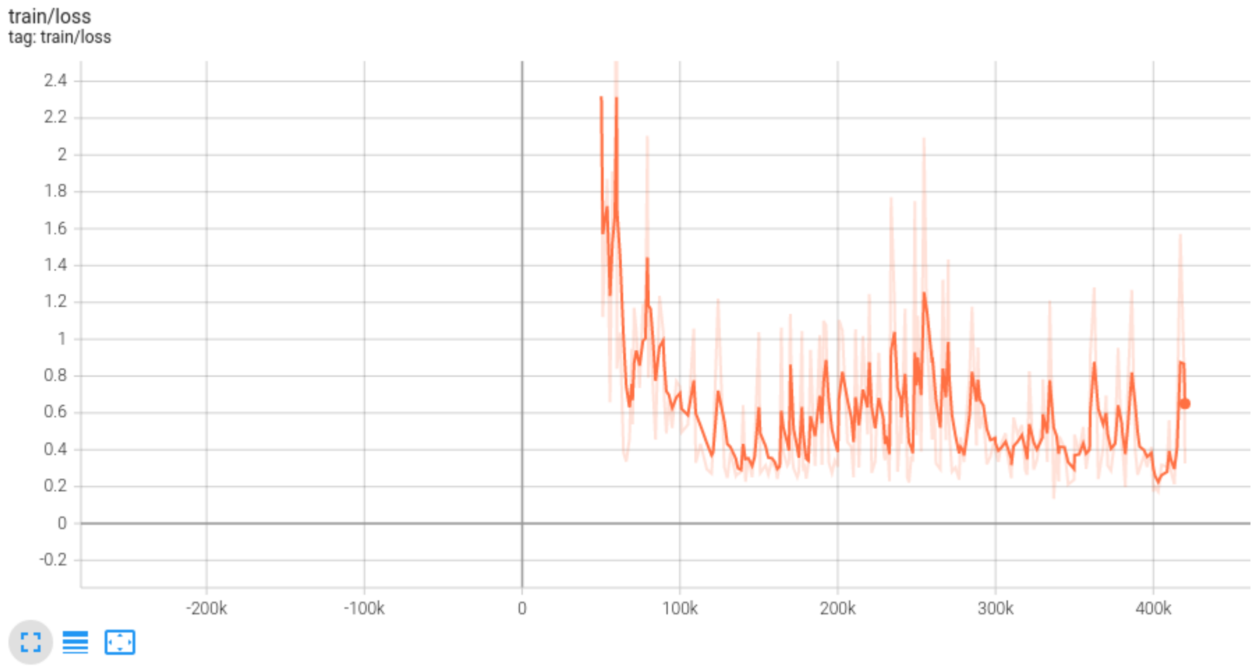


DQN

DQN is an off policy environment so it needs a lot of historical context to train. For DQN stablebaselines3 library was used, with recommended hyperparameters (from RL baselines zoo was used) to train the agent. The buffer size of the target network was kept at 50000, and the total training time steps were 10^5. To stop the training a criteria based on mean reward per 100 episodes was set. The threshold of the reward was kept to 150, and the training stopped just short of reaching the full timesteps. The training of DQN seemed jittery with oscillation between converging and diverging. It was overall slow to converge.

DQN Results:

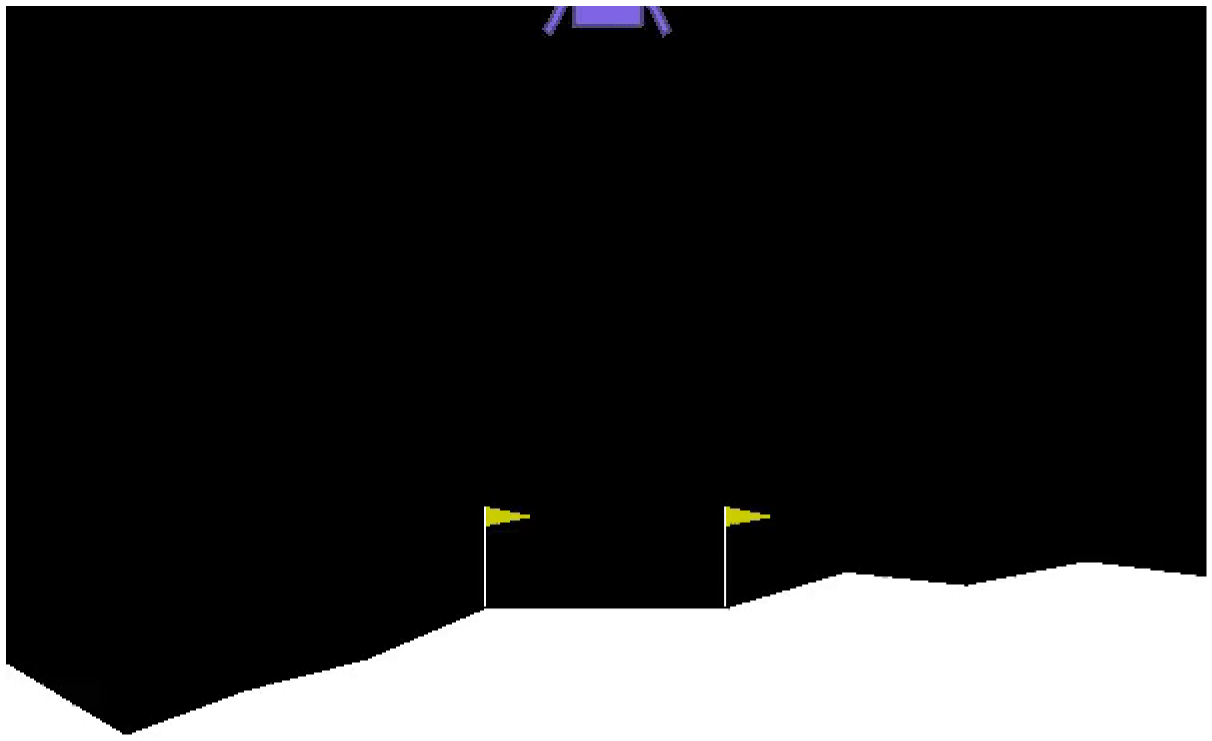


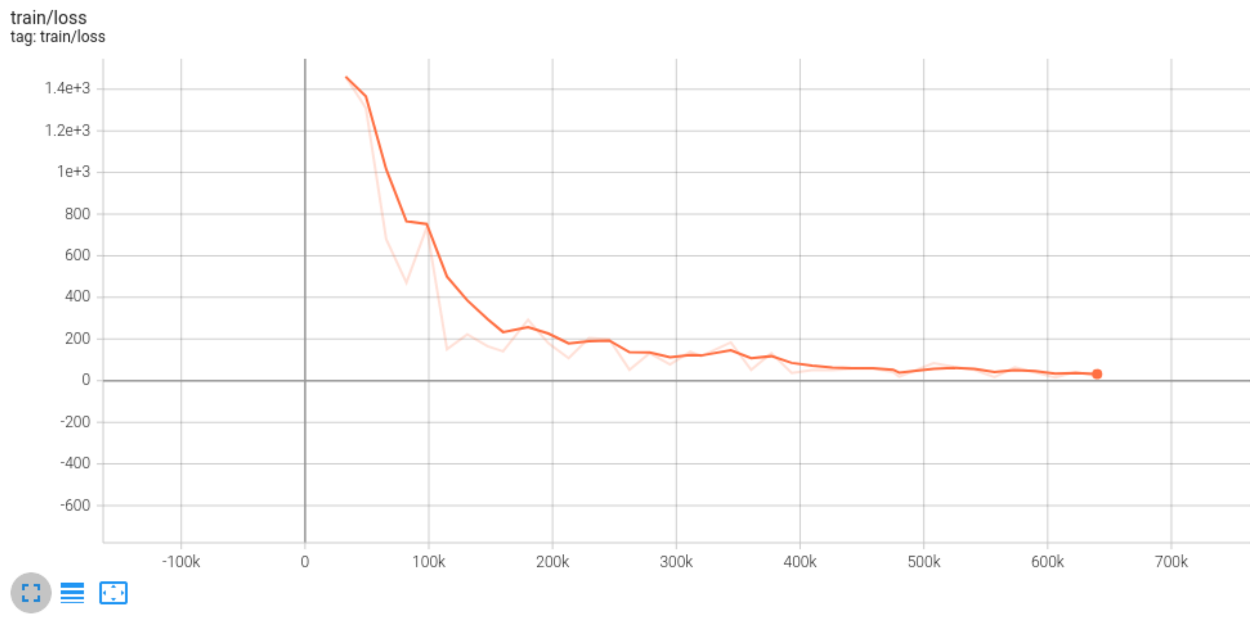

train/loss
tag: train/loss
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-200k
-100k
0
100k
200k
300k
400k

PPO

PPO from stablebaselines3 library was used to train the agent, with recommended parameters. The threshold to stop the training was set at 180. The training of the PPO happened in parallel with 16 environments, each having 1024 steps, and 4 epochs. The gamma was set to 0.999. PPO was fast to converge and was relatively more stable than DQN.

PPO Results:





Conclusions

For this environment SARSA worked surprisingly well however it needed the discretization of state space, and since the environment is not that complex, SARSA performed well. However for complex environments and interactions SARSA won't be suitable as it is easy for it to become intractable with increasing number of state spaces. DQN and PPO are more general purpose and a favourable choice for more complex environments. Between DQN and PPO, PPO worked better for this environment, it converged faster, and was relatively quicker to train, than DQN.

Appendix:

Project code ,reports and presentations are uploaded to a public repo on github.
https://github.com/Shashank-89/CMPE-260

References

[1] Gadgil, S., Xin, Y., & Xu, C. (2020). Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning. *arXiv [cs.LG]*. Opgehaald van http://arxiv.org/abs/2011.11850

[2] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym*.