Solving Lunar Lander with Reinforcement Learning

Shivani Bhakta

I. Introduction

Reinforcement Learning (RL) is a field of Machine training focused on the multi-stage decision making problem for an agent to perceive and interact with an environment, take some action and learn from the experiences in order to maximize some cumulative reward at the end. In recent years, RL has become more popular and has shown successful results in different application in Robotics. The classical RL methods used to be unstable for some applications when approximating a non-linear functions. To solve this, the use of Deep Neural Network (DNN) was introduced. In this paper, we will compare the performance of two different Deep RL algorithms on OpenAI's Gym environment - the lunar lander problem, and analyse the results. The first method used is Deep Q-Network (DQN) and second is Deep Deterministic Policy Gradient (DDPG).

II. Background

In the past few decades, humans' have made tremendous progress in it's space exploration advances. We have deployed satellites to Mars and are aiming to know about about other plants in our solar system. Thus, the need for safely landing the spaceship in different environment conditions and gravity also increases. Therefore, lunar lander problem has become more relevant and popular now than ever before. This is an important problem with many different applications. Apart from learning an optimal way to control the landing on different planet, we can use this on earth as we try to land our planes and other air vehicles in different weather conditions and different locations. In this paper, we aim to solve the lunar lander environment from the OpenAI gym using different RL methods and learn more about how they perform in different situation like the landing location and under lower gravity conditions. The goal is to make the agent land smoothly at the proper location between the flags, and use the fuel most efficiently.

There is lot of previous work done in solving the lunar lander problem using different techniques. However, there aren't many approaches proposed that takes into account for different uncertain conditions on the environment and on the agent. Therefore, in this project we want to not just look at the Deep RL algorithms and how they work on the environment, we want to test how well they work when different conditions and uncertainty is raised on the environment.

First, let us look at the model and the environment. We used the lunar lander v2 environment from the gym toolkit provided by the OpenAI for different RL applications. Previously, we have worked with other OpenAI environment, such as CartPole (Classic control), FrozenLake (Toy text) etc. LunarLander-v2 is part of their Box2D simulator.

The observation and state space is of length eight variables. These 8 variables are as follows:

- x coordinate of the lander's start position
- y coordinate of the lander's start position
- v_x Lander's velocity in x axis
- v_y Lander's velocity in y axis
- ω_θ Lander's angular velocity
- θ Lander's orientation
- $\{0,1\}$ Left leg on the ground
- $\{0,1\}$ Right leg on the ground

The action space consist of four discrete actions: $\{1, 2, 3, 4\}$ representing $\{$fire left engine orientation, fire right engine orientation, fire main engine, do nothing$\}$ respectively.

The following rewards are given based on the following conditions: Reward for moving from the top of the screen to the landing pad and zero speed is approximately 100-400 points. If lander crashes it's -100 points and if it lands it's +100 points. When either of the leg touched the ground lander gets +10 points. Once we reach more than 200 points, the environment is considered solved.

1) Reinforcement Learning: The RL problem is defined using Markov Decision Process (MDP). MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \pi, \gamma)$. Here $\mathcal{S}$ is a discrete/continuous set of states. $\mathcal{A}$ is a set of all actions, $\mathcal{R}$ are the rewards that the robot receoved for moving between different states via it's actions. π is the policy that maps all states to actions, γ is a discount factor $\in [0, 1]$

Value Functions V^π are defined as the expected return obtained after taking the actions given by the policy π starting at state s.

$$V^\pi(s) = \mathbb{E}_\pi[\mathbf{G}_t|\mathcal{S}_t = s] \tag{1}$$

Here

$$\mathbf{G}_t = \sum_{t=0}^{T} \gamma^t.R_t \tag{2}$$

return at each step t The policy π is used for finding the values for every state.

Action value Functions Q^π are also value functions that is defined as the expected return starting at state s, taking the action a, and then following the policy π.

$$Q^\pi(s, a) = \mathbb{E}_\pi[\mathbf{G}_t|\mathcal{S}_t = s, \mathcal{A}_t = a] \tag{3}$$

Thus, the goal of RL is to find the optimal policy, that gives us the highest reward. The optimal policy is determined using the optimal action value function $Q^*(s, a)$ which is obtained by using bellman equation.

Q-learning is the classic value based method in modern RL, to find the optimal Q^* values, which itetatively tells you the optimal policy π^*. This is done using the Q update:

$$Q^{\pi}(s,a) = Q^{\pi}(s,a) + \alpha(r(s,a) + \gamma \max_{a} Q(s^{'}, a^{'}) - Q^{\pi}(s,a)) \tag{4}$$

Q-Learning is a powerful algorithm for our agent to learn actions for all states. However, when the state space is large, it becomes difficult to use Q-Learning computationally. It could require a large Q matrix, which is hard to manage. Therefore, to solve this problem of computation expensive task, we use Neural Network.

In Deep Q-Networks we use a neural network to approximate the Q-value function.

III. Method

A. Deep Q-Learning

The deep Q-Learning algorithm uses multi-layer neural network called a Deep Q-Network (DQN) to estimate the Q-values function. The network takes the states as the input (8 dimensional state for Lunar Lander) and generates the Q-values for all possible actions in the output. The Q-value update rule remains the same as Q-Learning:

$$Q^{\pi}(s,a) = Q^{\pi}(s,a) + \alpha(r(s,a) + \gamma \max_{a} Q(s^{'}, a^{'}) - Q^{\pi}(s,a)) \tag{5}$$

However, the optimal $Q^*(s,a)$ is calculated using the deep neural network with parameter θ. DQN uses an experience replay. Basically, at each time stamp t, values of current state, next state, reward and action is store into the replay memory. We then randomly sample a mini-batch of samples from memory and use it to train our network model. We mainly use the replay memory because the consecutive samples are closely related and we want to obtain uncorrelated sample for more accurate gradient descent estimate.

This is a regression problem, thus we use the following Loss function:

$$L(\theta) = 1/n \sum_{i=1}^{n} [Y_i - Q_{\theta}(s_i, a_i)]^2 \tag{6}$$

where

$Y_i = r + \gamma \max_a \max_{a_\theta} Q_{\theta}(s^{'}, a^{'})$

For our neural network, we use a 3 layer neural network. We use ReLU activation for the hidden layers and Linear activation for the output layer. Following parameters were used to train this network, learning rate = 0.001, batch size = 64, gamma = 0.99 (discount factor), epsilon start from 1 to 0.001 with decay of 0.999, num of steps taken per episodes are 1000. The following is the DQN algorithm used for our project.

Algorithm 1 Deep Q-Learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ.
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for *episode = 1,M* **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for *t=1,T* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j \text{ if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) \quad \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end
end

As it can be seen in the algorithm, we used ϵ-greedy policy to select the action for the given state, using the epsilon decay discussed above.

To reconsider, the main goal of the problem is to direct the lunar lander to the stable landing position (between the two poles) softly, while saving the fuel, as quickly as possible with both legs on the ground. Once the lunar lander has either crashed or landed, we end the algorithm and it will not move. In this environment, firing left or the right engine leads the lander to spin in different direction and different forces causes it to go in random direction that it should go to. We would want to resolve this using our two algorithms DQN and DDPG.

We would also like to observe how these algorithm performs as we change different uncertainty. Due to time constraint, we didn't get to perform these task, but we will describe what we would do to test these.

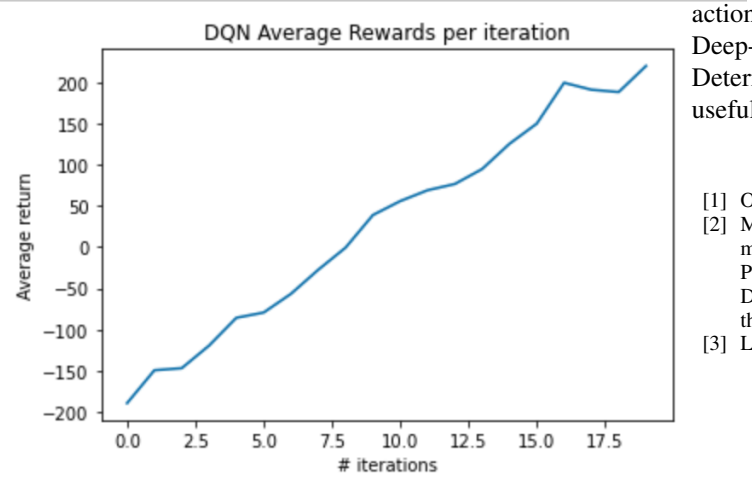
After solving the original problem, we will introduce additional uncertainty like noisy observations of the agent's state, unleveled surface, unclear flag pole locations, gravity changes, random torque applied, random engine failure etc. We will then analyse the performance of our algorithms relative to each other as well as relative to each change of conditions.

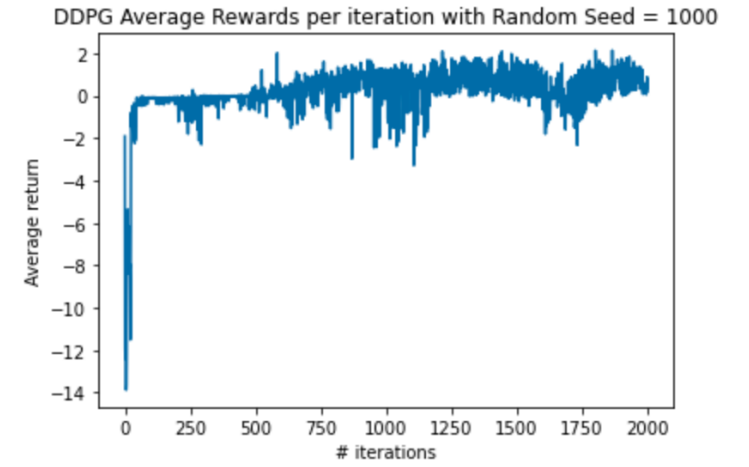
B. Deep Deterministic Policy Gradient

DDPG is a model free off-policy algorithm which we used on the continuous Lunar Lander environment. This algorithm is a combination of DQN and Deterministic Policy Gradient

IV. Results

The figures shows the average reward obtained every 100 episodes by DQN agent. At the beginning the reward is negative, this is when the agent is exploring and learning about





each state transitions. Once most of the map is explored the Q-values start to converge. DQN Agent for the Lunar Lander environment solved the environment (trained) in around 1800 episodes.

As can be seen in the plot, both algorithm works really well for the regular environment.However DQN coverges faster and than DDPG and takes up less time to train as well.

For this project I really wanted to learn more about how these algorithm perform in different uncertain environment condition but due to lack of time, and because I spent countless hours on different algorithms like Deep SARSA, soft actor critic and other deep RL that I couldn't get to work. With more time I would like to test these algorithms with different conditions as well as propose any potential changes to these algorithm to make them perform better. This will provide a great overview of how robust each algorithm is considering all the uncertainties.

Conclusion

Regular Q-Learning algorithms is very powerful for training a Reinforcement Learning Agent as long as the state and action spaces are finite. However, when we have large space Deep-Learning Algorithms such as Deep Q-Network, Deep Deterministic Policy Gradient or Double Deep Q-Learning are useful and will give us great results.

References

[1] OpenAI

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540):529–533.

[3] Lecture Slides ECE 276C UC San Diego