Index

# Introduction

## Problem statement: Lunar Lander V2

**Goal:** Land on the landing pad with zero speed and least fuel consumption. Achieve 200 points

### Rewards:

### Winning the game is 200 points

### Reward for moving from the top of the screen to landing pad and zero speed is about 100 to 140 points.

* If lander moves away from landing pad it loses reward back.
* Episode finishes if the lander crashes -100 points or comes to rest +100 points
* Each leg ground contact is +10
* Firing main engine is -0.3 points each frame

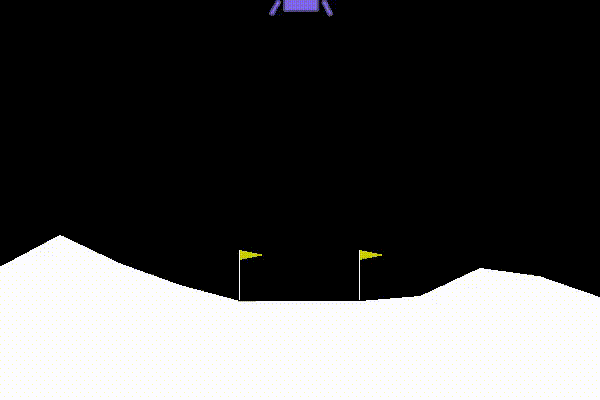
### Action Space:

Four discrete actions available,

1. Do nothing
2. Fire left orientation engine
3. Fire main engine
4. fire right orientation engine

Note: Landing outside landing pad is possible. Fuel is infinite

Sample run below from the Open AI Webpage



.

## Objective:

Comparisons of how different Model Free Reinforcement Learning algorithms stack against each other when running against the Open AI Gym Lunar lander environment. Comparison is based on factual data (rewards obtained) over thousands of episodes for each algorithm.

## Scope:

Following algorithms are chosen for this experiment, keeping track of continuous states and model free problem, on policy algorithms, faster q value convergence along with considering factors like epsilon greedy-epsilon soft, learning rate and discounted future rewards etc.

1. Monte Carlo Control (On-Policy without Exploring starts)

2. SARSA (TD On-Policy)

3. Q Learning (TD Off-Policy)

4. Double Q (TD Off-Policy)

5. Expected SARSA (TD Off-Policy)

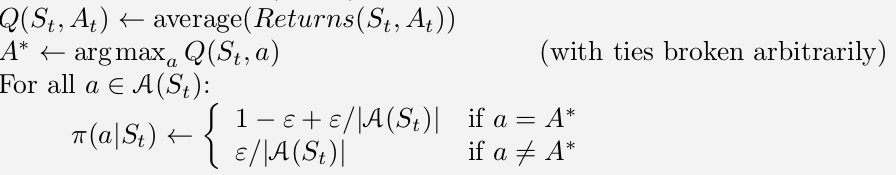
6. Deep Q Network

Other algorithms like Dynamic Programming, Monte Carlo Exploration starts and Monte Carlo off policy are kept out of scope from this experiment.

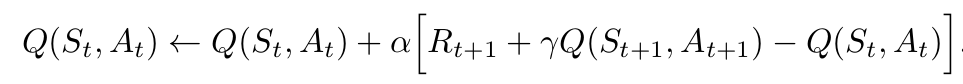
# Brief on Algorithms

1. **Monte Carlo (On-Policy without Exploring starts):** On-policy methods attempt to evaluate or improve the policy that is used to make decisions. that does not use the unrealistic assumption of exploring starts. the policy is generally soft, meaning that pi|(s) > 0 for all states and all actions, but gradually shifted closer and closer to a deterministic optimal policy.

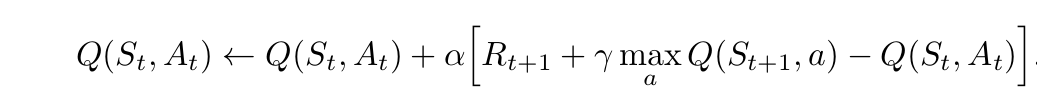




1. **SARSA:** This is on-policy TD control algorithm. we continually estimate q(pi) for the behaviour policy (pi), and at the same time change (pi) toward greediness with respect to q(pi). The convergence properties of the Sarsa algorithm depend on the nature of the policy’s dependence on Q. For example, one could use "-greedy or "-soft policies. Sarsa converges with probability 1 to an optimal policy and action-value function as long as all state–action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy.

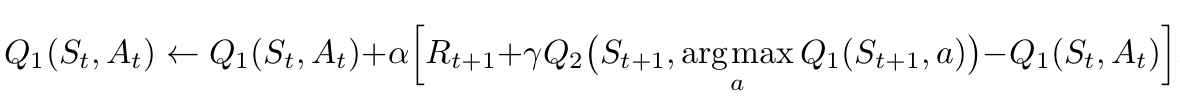


1. **Q Learning:** This is one of the off-policy TD control algorithm. In this algorithm the learned action-value function, Q, directly approximates q\*, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has an effect in that it determines which state–action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated.



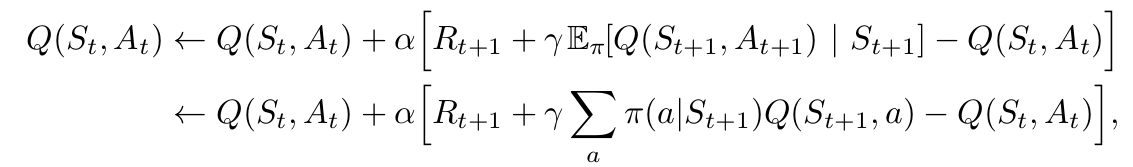
1. **Double Q Learning:** Under Temporal Difference Algorithms, Double Q is another algorithm which is mainly used over other learning algorithms to avoid a positive maximization bias if we use the maximum of the estimates as an estimate of the maximum of the true values. One way to view the problem is that it is due to using the same samples (plays) both to determine the maximizing action and to estimate its value. Hence as a solution, use one estimate, say Q1 , to determine the maximizing action A\* = argmax a Q 1(a), and the other, Q2 , to provide the estimate of its value, Q 2 (A\*) = Q2(argmax a Q 1(a)). This estimate will then be unbiased in the sense that E[Q2(A\*)] = q(A\*). We can also repeat the process with the role of the two estimates reversed to yield a second unbiased estimate Q 1 (argmax a Q2(a)). This is the idea of double learning. Note that although we learn two estimates, only one estimate is updated on each play; double learning doubles the memory requirements, but does not increase the amount of computation per step

Update Rule:-



1. **Expected SARSA:** Under Temporal Difference Algorithms, Expected SARSA is one of the faster convergence algorithms where how likely each action is doing under the current policy is also considered by using its probability (For action probability distribution, epsilon soft poilicy is being used in the algorithm). This accounts for expected value of each next state, action pair and used directly in the update rule of the alogorithm.

Update Rule:-

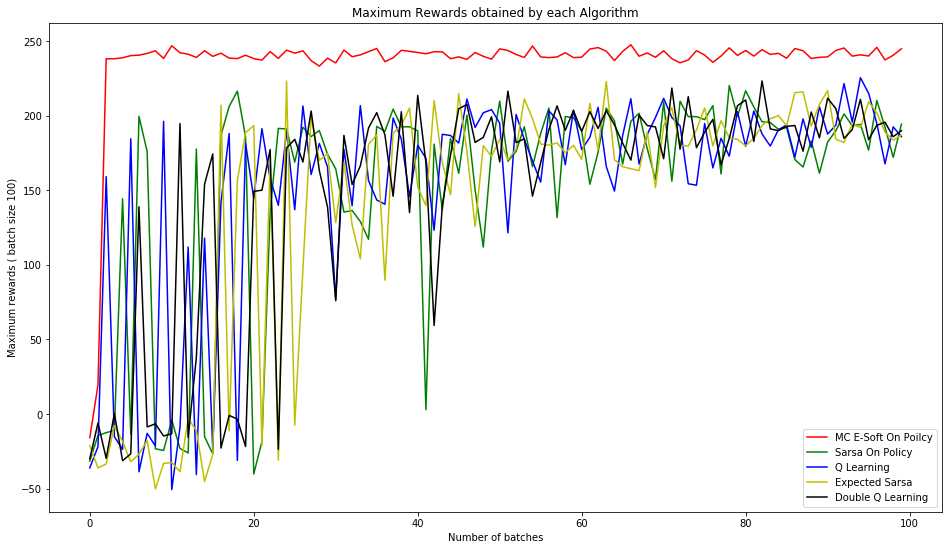
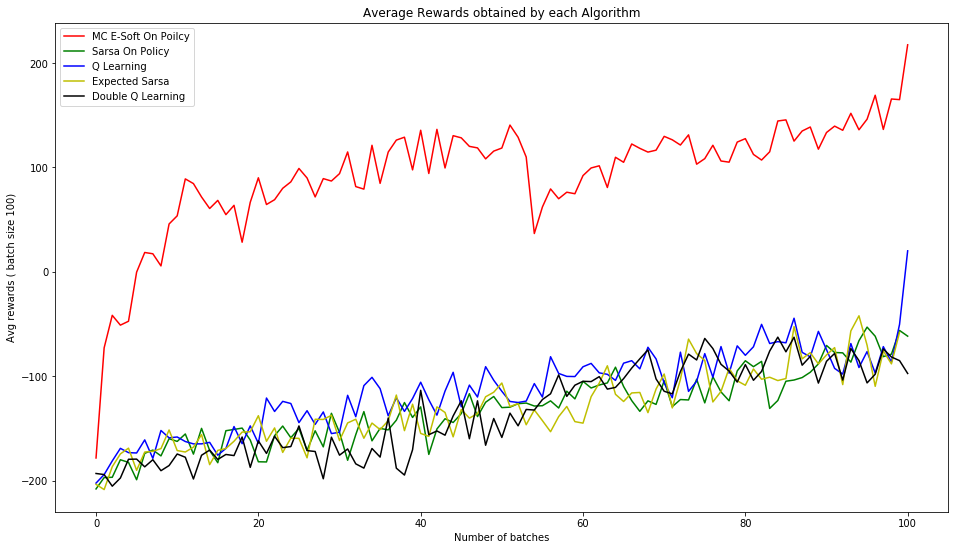


# Results

We ran the each of the algorithms mentioned earlier for 10,000 episodes on the Lunar lander environment.

To ensure fair comparison between each of the algorithms the algorithm parameter like Learning Rate (0.1), Discount Factor (0.95) and Epsilon (0.1) are retained as same across different algorithms, Also to the environment is seeded to ensure consistency.

Below graphs display Average rewards (batch size 100 episodes) and Maximum rewards (batch size 100 episodes)



Monte Carlo On Policy (E Soft) algorithm did perform better then all the Temporal Difference algorithms in both average rewards and the maximum rewards obtained.

The performance of the Temporal Difference algorithms is very similar amongst themselves when it comes to rewards obtained.

There is a lot of variance even after 10000 episodes in the TD algorithms The average rewards is -100, where as corresponding maximum reward is in range 150 to 200.

To understand the difference the learning rate makes in the average / maximum rewar

Continuing the experiment for more episodes would have helped to to reduce this variance and obtain higher reward.

Below are videos for sample episodes for each of the algorithms. If any issues are observed in playing embedded videos, These are also available in the attached package.

|  |  |
| --- | --- |
| Monte Carlo On Policy E Soft |  |

|  |  |
| --- | --- |
| Sarsa | Expected Sarsa |

|  |  |
| --- | --- |
| Q Learning | Double Q Learning |

# Solution Packaging :

Entire experiment is packaged under a folder – Project\_LunarLander

There are three sub folders inside– Programs , PKLFiles, Reports\_ VideoClips

Programs - It contains programs of different algorithms.

a) spylander\_monte\_carlo\_every\_visit.py

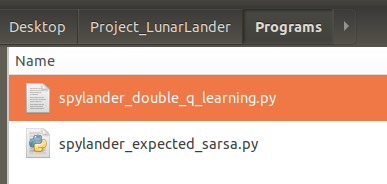
b) spylander\_sarsa.py

c) spylander\_q\_learning.py

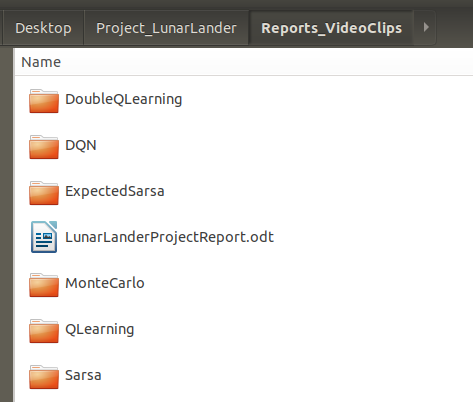
d) spylander\_double\_q\_learning.py

e) spylander\_expected\_sarsa.py

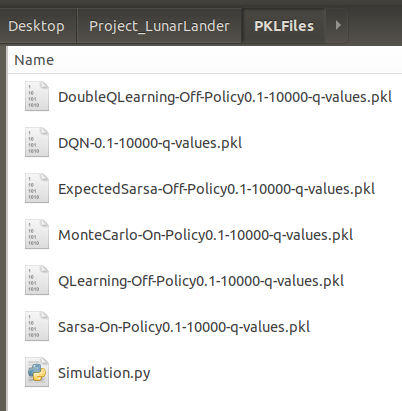
f) DQN.py



Reports\_VideoClips – It contains Project Report and some Video Clips of different algorithms.



PKLFiles – It contains PKLFiles of different algorithms and a program “Simulation.py” which helps in running the experiment against each algorithm. It is explained in “How to run the experiment” section below.



## **Tech Stack:**

### OS Requirement: Ubuntu Linux 18.04 / Mac OS (Windows is not recommended)

### Python: Version 3.6 / 3.7 (Note Open AI does not support more recent versions of python)

### Hardware requirement:

### All algorithms except for DQN, Recommended 16 GB of ram to run simultaneous simulations.

DQN ??

### Dependencies / Libraries:

Open AI gym: Version 0.15

Install all the environments (including “Lunar lander v2”). For instructions on how to install Open AI gym refer [https://github.com/openai/gym#installing-everything](https://github.com/openai/gym" \l "installing-everything).

Open AI Gym Monitor plugin is used for collecting stats like rewards per episode and also to capture videos of the simulations below dependencies should be installed.

xvfb (X11 Virtual frame buffer)

ffmpef

Installation commands for these dependencies

sudo apt-get install xvfb

sudo apt install ffmpeg

# How to run the experiment:

Open the terminal in ubuntu system and execute the below command:-

**python simulation.py arg1 (<path of .pkl file>) arg2 (<number of episodes>)**

### simulation.py: This script is created for running the experiment using \*.pkl file which is generated during the training phase over thousands of episodes and contains q values for all state-action pairs. In this simulated run, user can directly run the experiment by passing learnt q values.

### arg1 (<path of \*.pkl file>): User has to provide the full path of .pkl file for the particular algorithm he wants to view.

### **agr2 (number of episodes):** User has to provide the number of iterations he wants to see the experiment for a particular algorithm.

User should see Lunar Lander’s trained model with higher rates of successful landing on the seeded environment.