Train Your Lunar-Lander | Reinforcement Learning

Shiva Verma Follow
Apr 20 · 6 min read ★

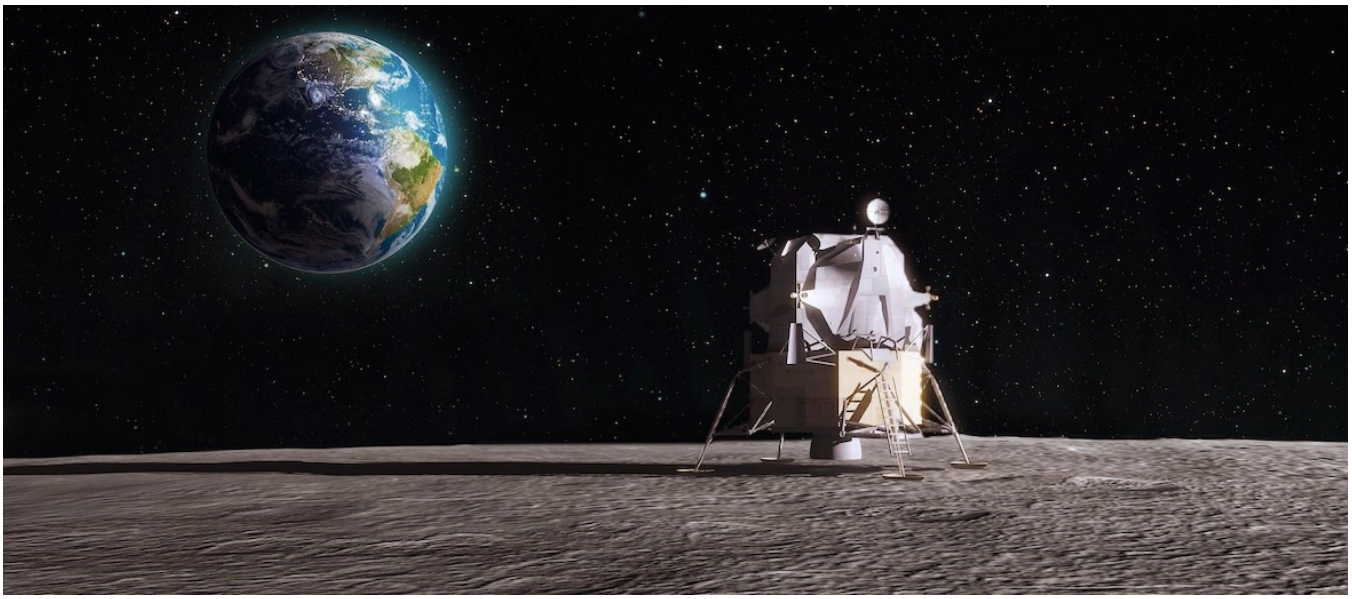
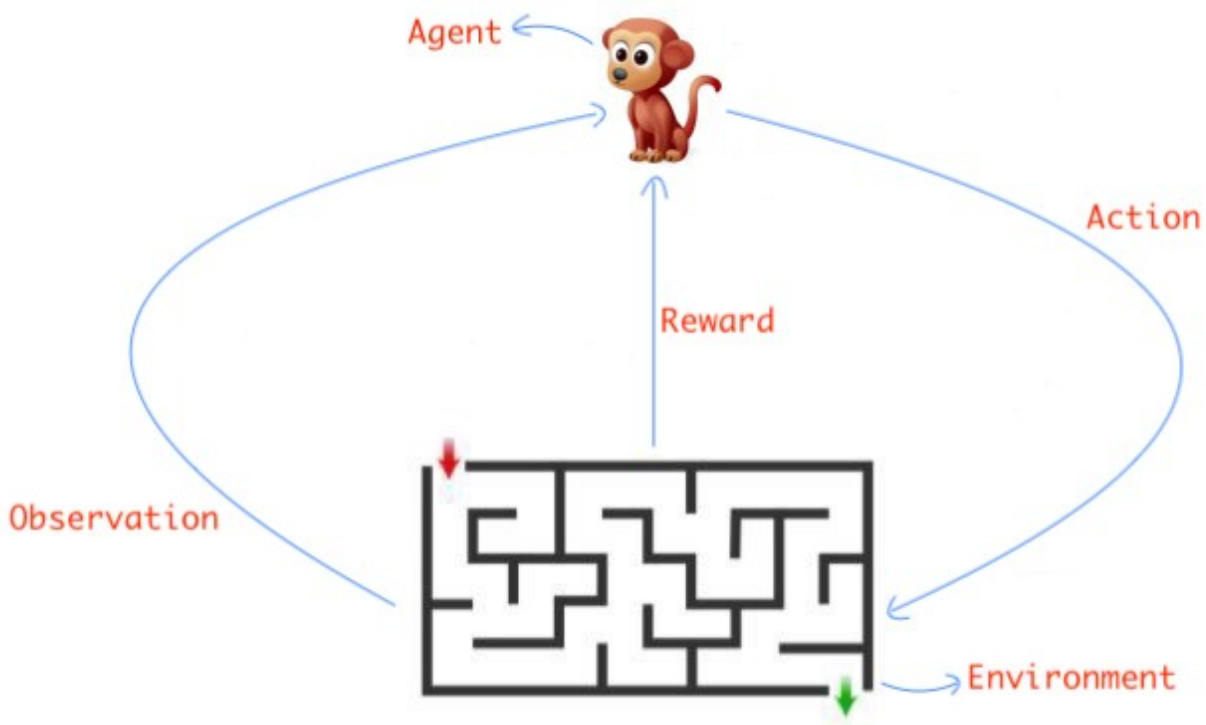


Image by AlphaCoders

Lunar Lander is another interesting problem in OpenAIGym. In my previous blog, I solved the classic control environments. In this blog, I will be solving the Lunar Lander environment.

Reinforcement Learning | Brief Intro

Reinforcement learning is an interesting area of Machine learning. The rough Idea is that you have an **agent** and an **environment**. The agent takes actions and environment gives reward based on those actions, The goal is to teach the agent optimal behaviour in order to maximize the reward received by the environment.



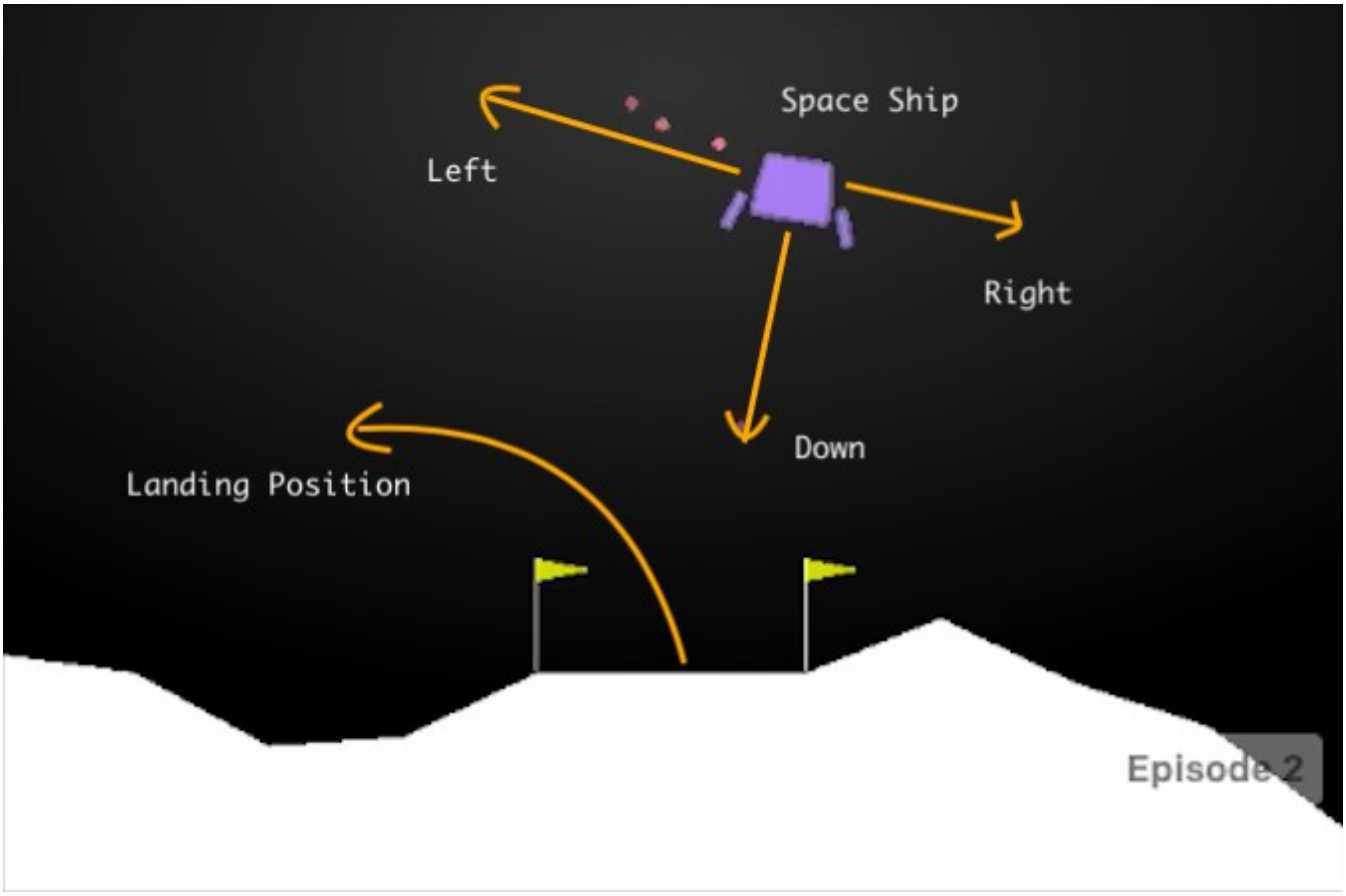
Reinforcement Learning Diagram

For example, have a look at the diagram. This maze represents our **environment**. Our purpose would be to teach the agent an optimal policy so that it can solve this maze. The maze will provide a reward to the agent based on the goodness of each action it takes. Also, each action taken by agent leads it to the new **state** in the environment.

About Lunar-Lander

As you can see in the picture below, there is one space-ship. The task is to

land the space-ship between the flags smoothly. The ship has 3 throttles in it. One throttle points downward and other 2 points in the left and right direction. With the help of these, you have to control the Ship.



There are 2 different Lunar Lander Environment in OpenAIGym. One has discrete action space and other has continuous action space. Let's solve both one by one. Please read **this** doc to know how to use Gym environments.

LunarLander-v2 (Discrete)

> *Landing pad is always at coordinates (0,0). Coordinates are the first two numbers in state vector. Reward for moving from the top of the screen to*

> *landing pad and zero speed is about 100..140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10. Firing main engine is -0.3 points each frame. Solved is 200 points. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. Four discrete actions available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.*

This quote provides enough details about the action and state space. Also, I observed that there is some random wind in the environment which influence the direction of the Ship.

Action space (Discrete)

- `0` - Do nothing
- `1` - Fire left engine
- `2` - Fire down engine
- `3` - Fire right engine

Solving the environment

I am solving this problem with the DQN algorithm, which is compatible and works well when you have a discrete action space and continuous state space.

I will not be going into details of how DQN works. DQN approximate the actions using a neural network. There is much more to read about it. There are pretty good resources on the DQN online. I have included the link of

these resources at the end of this blog.

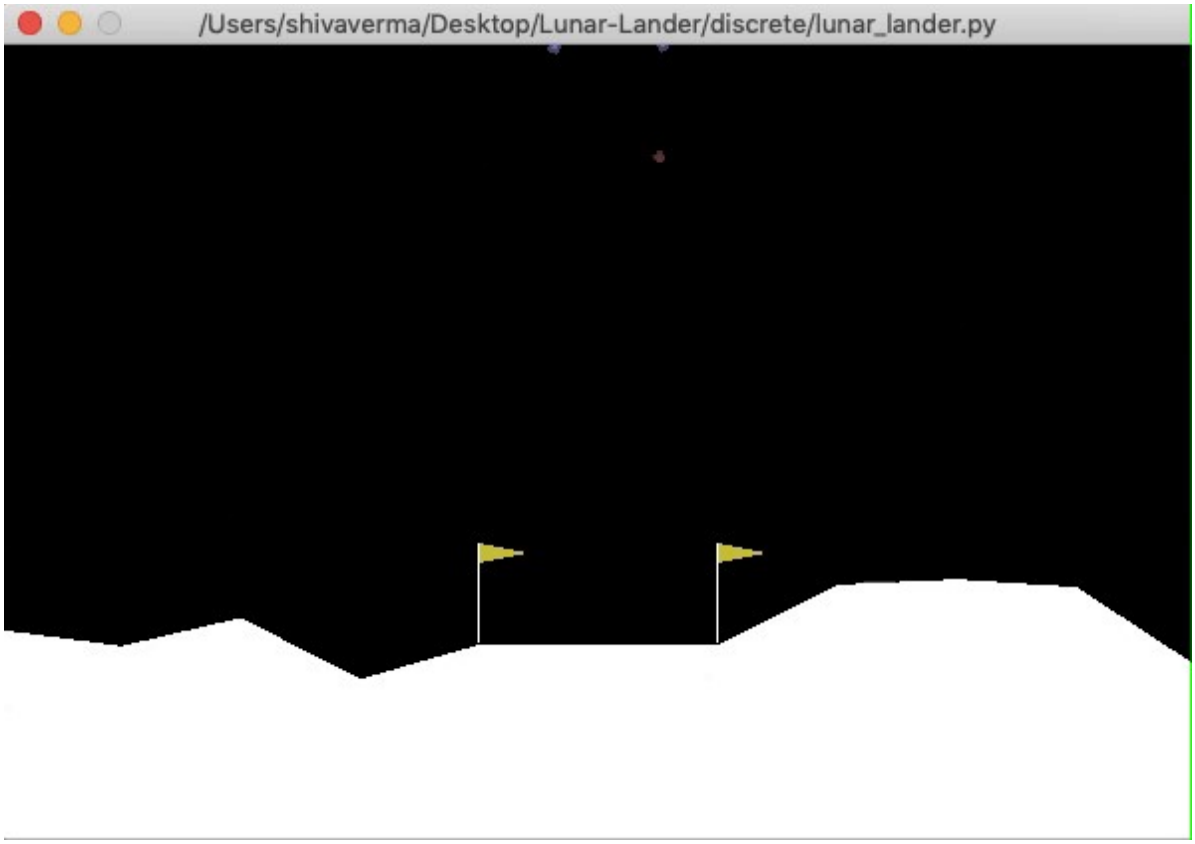
Network Architecture

I have attached the snippet of my DQN algorithm which shows network architecture and hyperparameters I have used.

Input size of the network should be equal to the number of states. Output size of the network should be equal to the number of actions an agent can take. If there are 4 possible actions then the network will output 4 scores. These 4 scores correspond to 4 actions and we select the action which has the highest score.

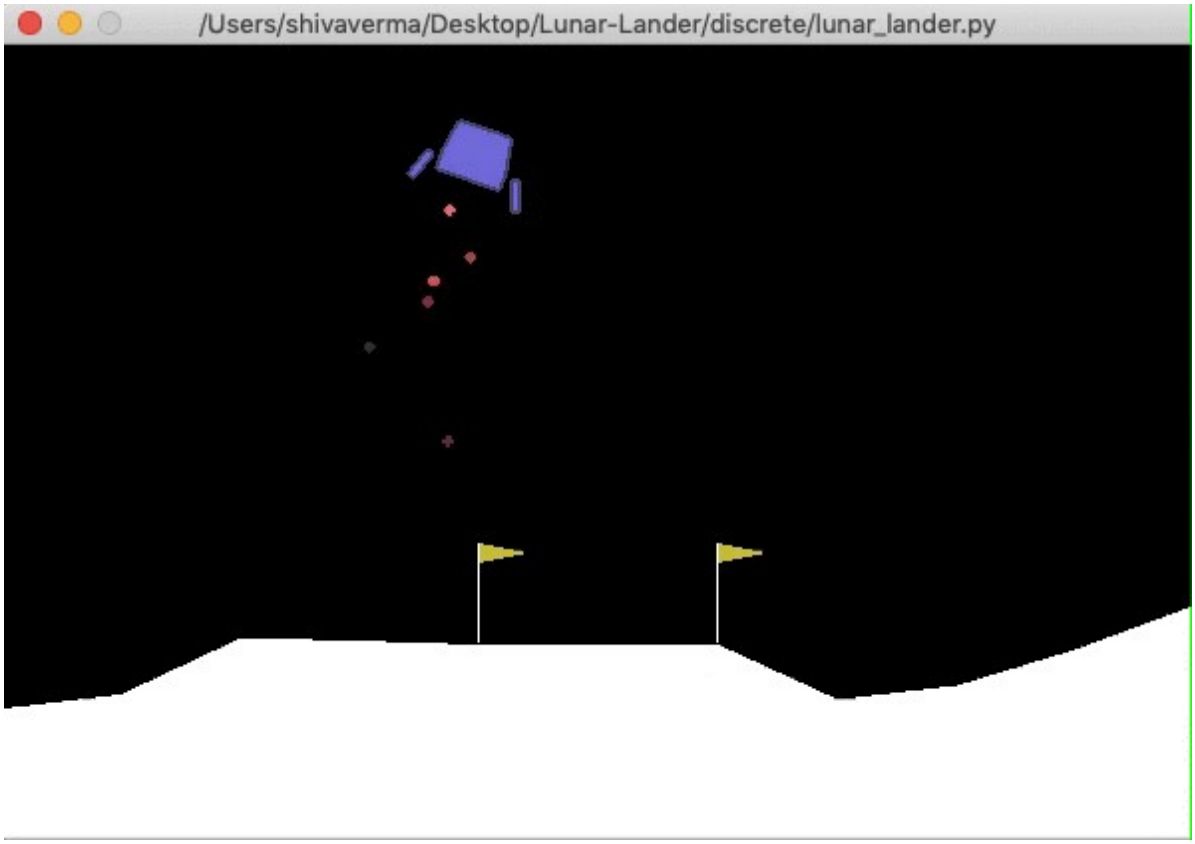
Training Visualization

- Following gifs shows the learning progress. In starting the agent is behaving very poorly. It does not know how to control the throttles.



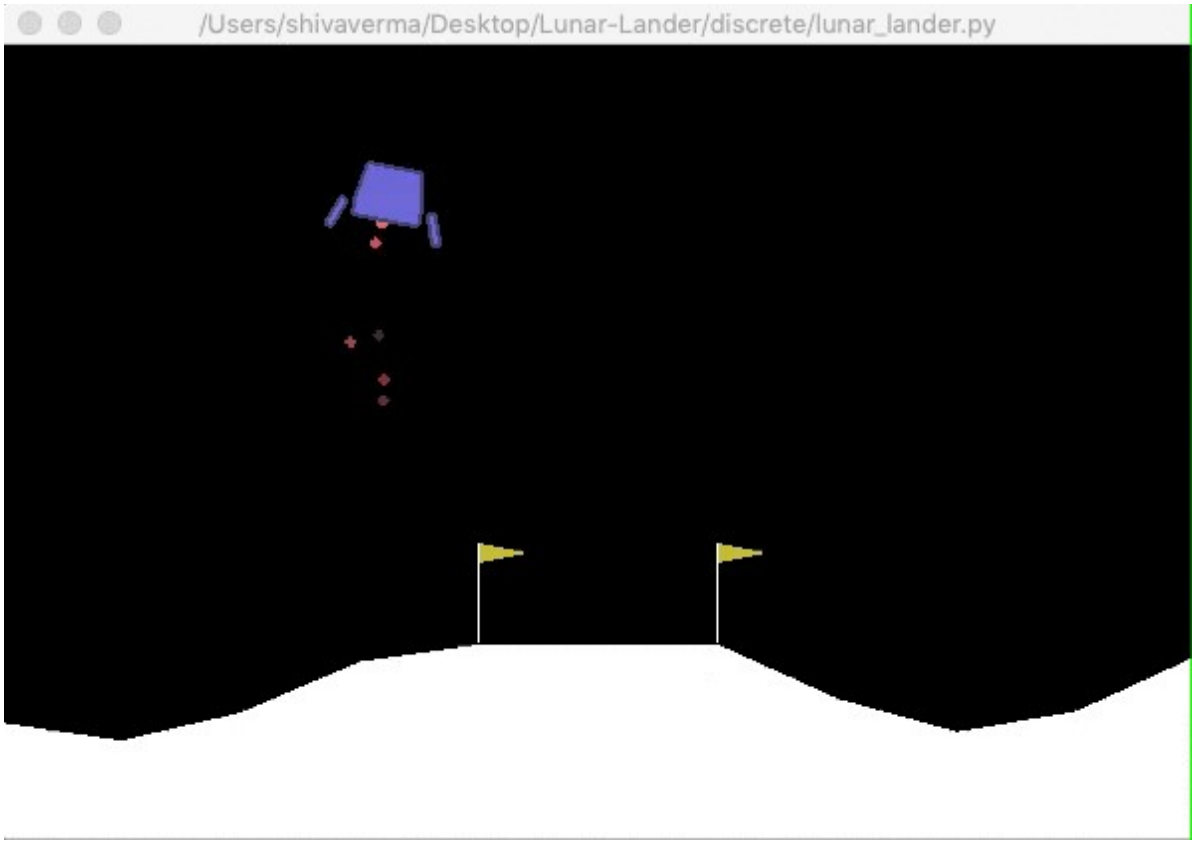
Before training

- After 50 episodes ship started to have better control of the throttles.



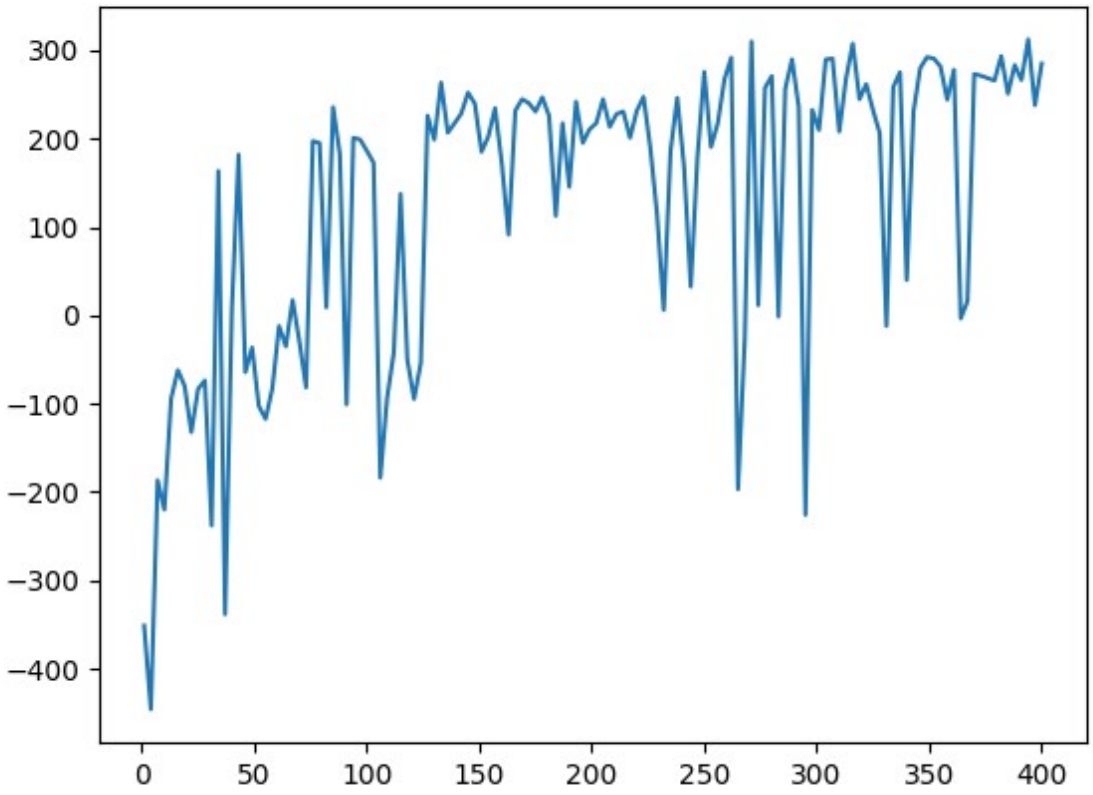
Middle of the training

- After around 120 episodes, ship masters the task and started to land very smoothly every single time.



After complete training

Following is the plot showing rewards per episode. In order to solve the environment, you have to achieve an average reward of +200 for 100 consecutive episodes. I was able to solve the environment in around 400 episodes.



Reward vs Episode plot (discrete)

. . .

LunarLanderContinuous-v2 (Continuous)

> *Landing pad is always at coordinates (0,0). Coordinates are the first two numbers in state vector. Reward for moving from the top of the screen to landing pad and zero speed is about 100..140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10. Firing main engine is -0.3 points each frame. Solved is 200 points. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. Action is two real values*

> *vector from -1 to +1. First controls main engine, -1..0 off, 0..+1 throttle from 50% to 100% power. Engine can't work with less than 50% power. Second value -1.0..-0.5 fire left engine, +0.5..+1.0 fire right engine, -0.5..0.5 off.*

Solving the environment

This problem is slightly different. Action space is continuous here. I am using the DDPG algorithm to solve this problem. DDPG works quite well when we have continuous state and state space.

In DDPG there are two networks called Actor and Critic. Actor-network output action value, given states to it. Critic network output the Q value (how good state-action pair is), given state and action(produces to by the actor-network) value pair. You can read about the DDPG in detail from the sources available online. I have also attached some link in the end.

Actor-Network Architecture

Following is the code snippet of my actor-network. My network architecture consists of 2 hidden layers with **batch-norm** and **relu** activation. Size of the hidden layers is 400 and 200. I am using **tanh** activation on final layer because the action is bound in the range (-1, 1). Instead of using default weights, I initialized my weights which improved the performance.

Actor-Network

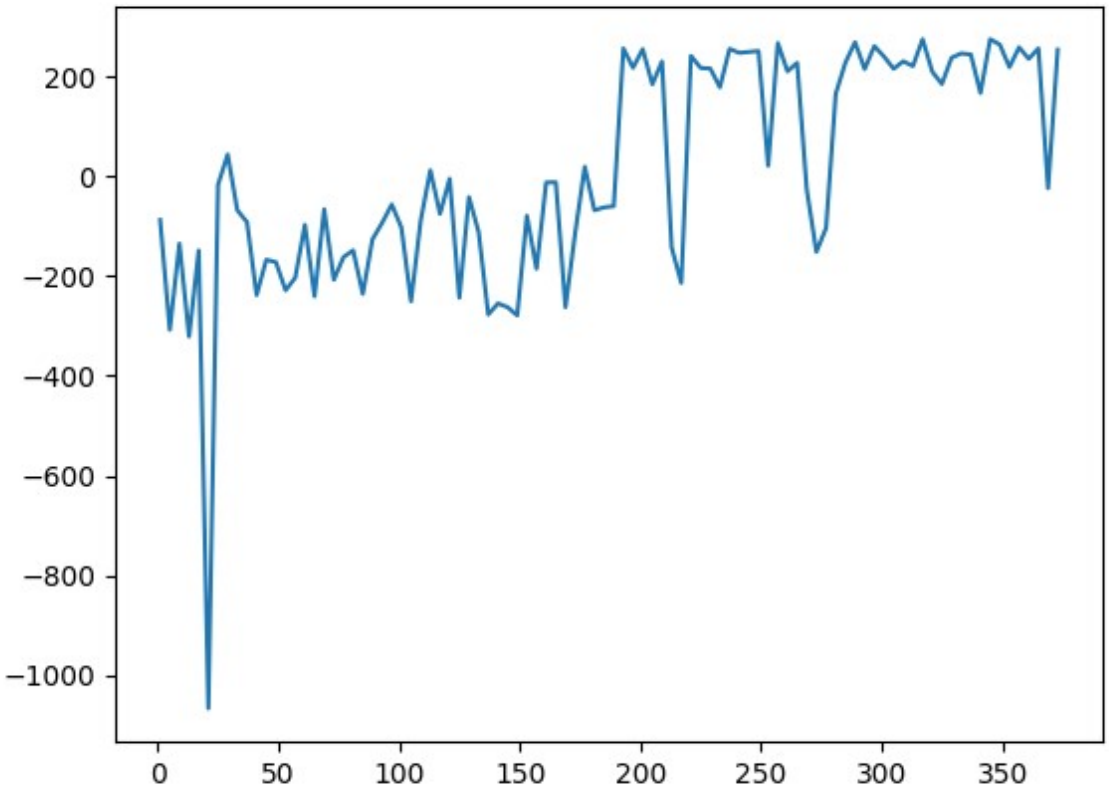
Critic-Network Architecture

Following is the code snippet of my critic-network. There are 2 hidden layers for state and 1 hidden state for action. I am using **batch-norm** and

relu activation here. Hidden layers of action and space are added together with **relu** activation on it. There is no activation on final layer because it's output is a **Q-value.**

Critic-Network

Following is the plot showing rewards per episode. I was able to solve the task in around 350 episodes.



Reward vs Episode Plot (continuous)

. . .

Resources

- This is an awesome introductory **blog** on Reinforcement Learning.
- I will highly recommend you to read the **paper** on DQN by Deepmind.
- Here is the **paper** on DDPG.
- This is a very helpful **blog** on DDPG.
- Read **this** doc to know how to use Gym environments.
- Check out other cool **environments** on OpenAIGym.
- This is my previous **blog** on OpenAIGym Classic control problems.
- Below is the link to my GitHub repository for this project.

shivaverma/OpenAIGym

Solving OpenAI Gym problems. Contribute to shivaverma/OpenAIGym development by creating an account on...

github.com

Machine Learning OpenAI Reinforcement Learning Deep Learning Artificial Intelligence