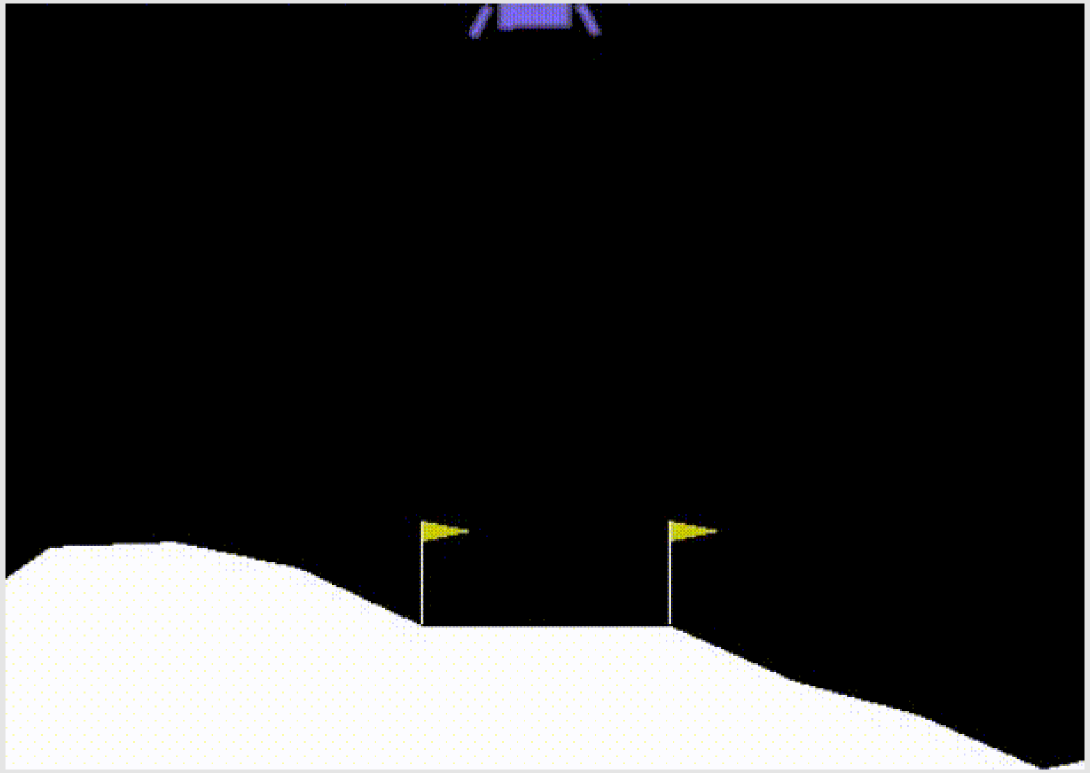
# Lunar Lander problem using a Deep Q-learning Neural Network

2021 Fall Hack-A-Roo

Srinivas Rahul Sapireddy

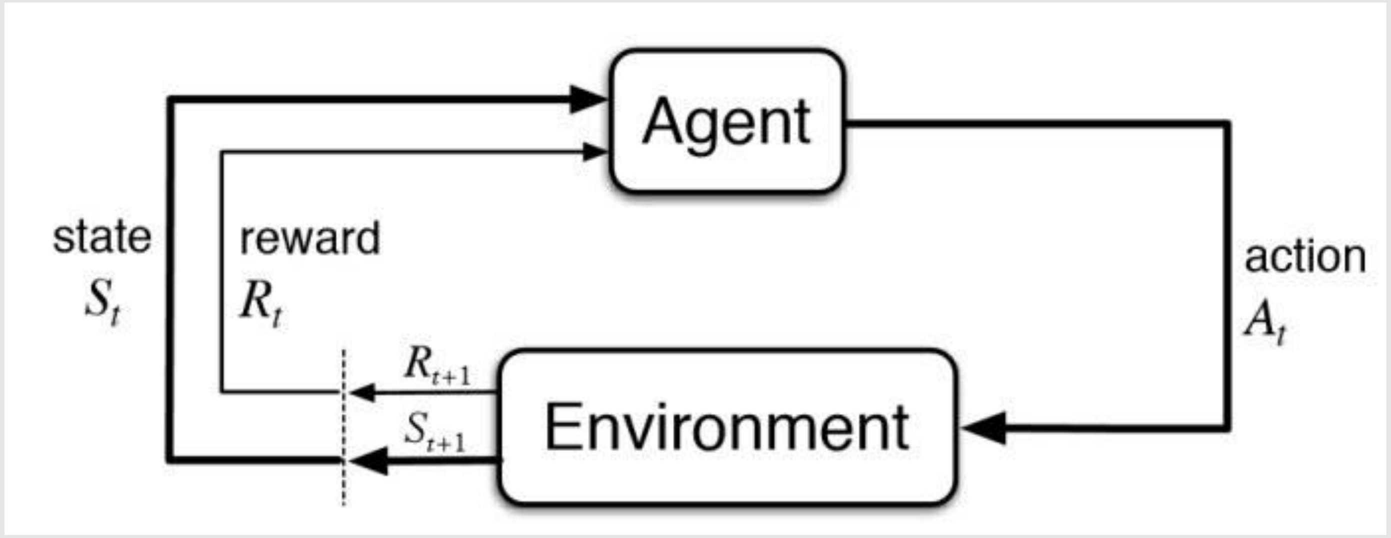
UMKC
School of Computing and Engineering

## Problem Statement



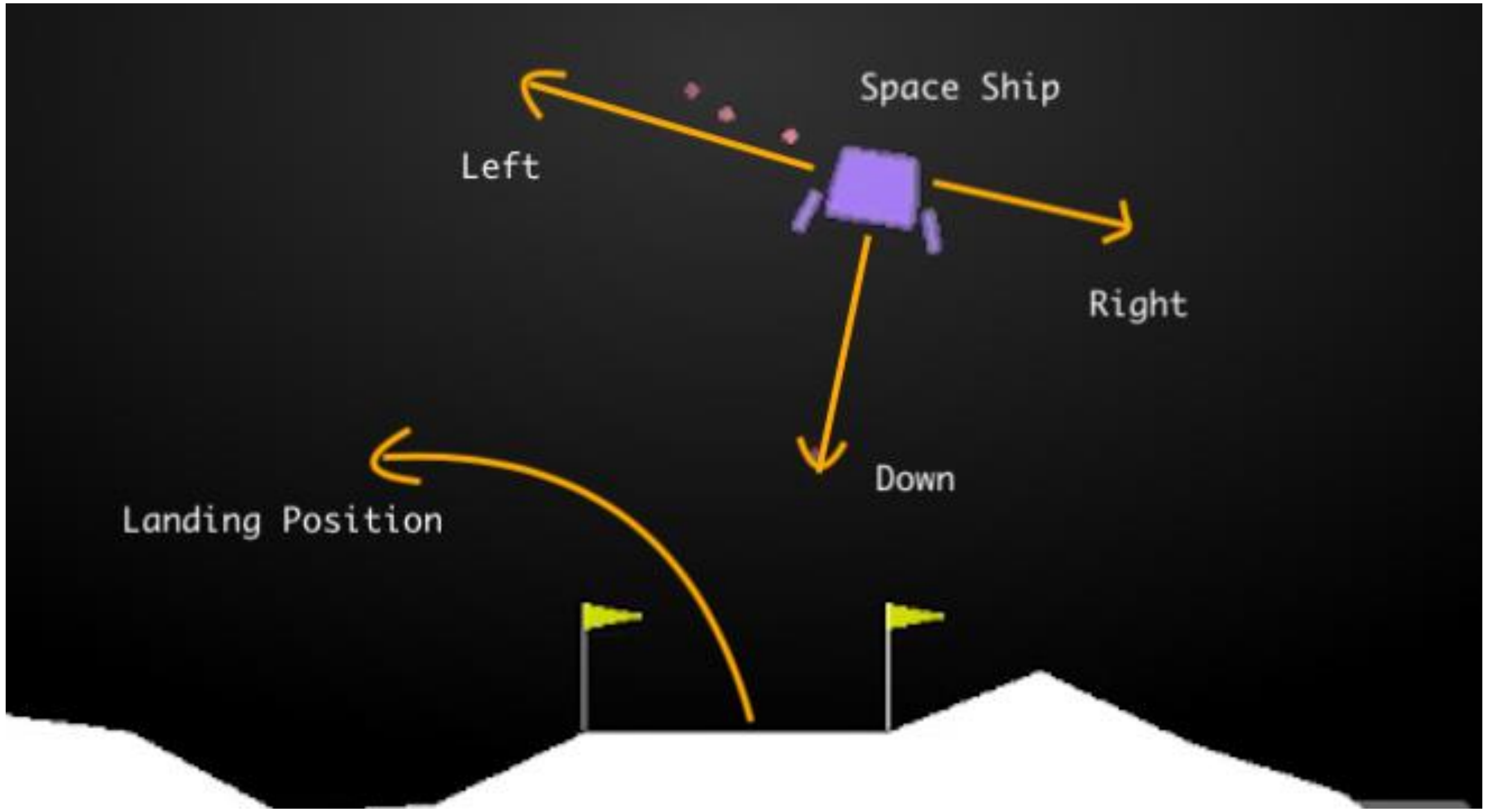
Solving Lunar Lander problem provided by OpenAI Gym using Deep Q Learning Neural Network.

# DQN Algorithm



- Deep Q learning neural network is used to calculate the required action at each step in Open AI Gym environment.
- Observation space: Coordinates
- Actions: The 4 actions taken by the lunar lander in each frame or taking no action at all.
- Also use Double DQN to evaluate the original network. Solves the overestimation of action.

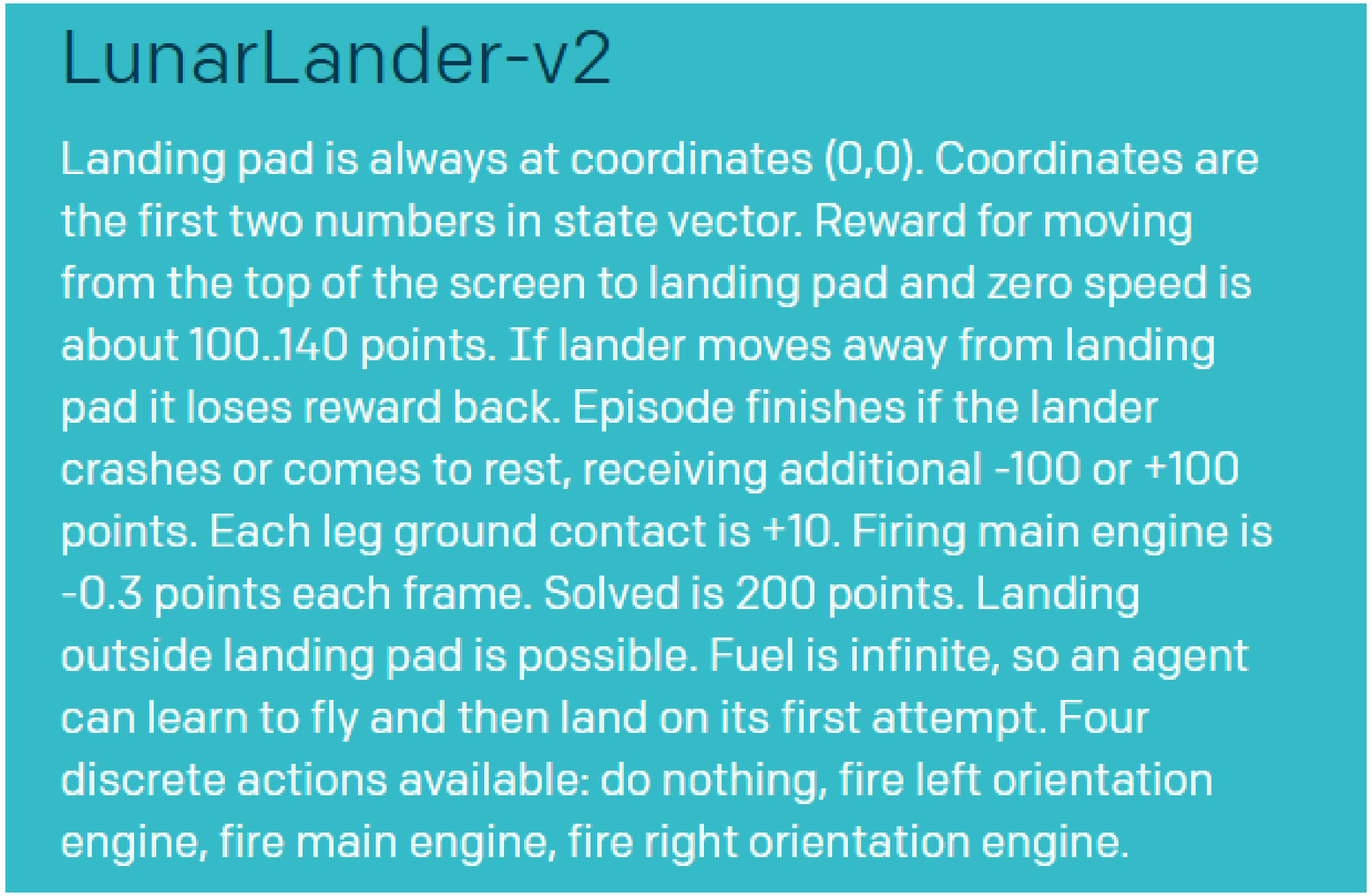
# Action Space



## Data

- Generate data using the environment for simulation.
- Observation space includes coordinates of the lander.
- Action space includes 4 actions.

Environment Source:
[https://gym.openai.com/envs/LunarLander-v2/](https://gym.openai.com/envs/LunarLander-v2/)



### LunarLander-v2

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.

## Code and Tools's Used

Requires Python 3 or above.

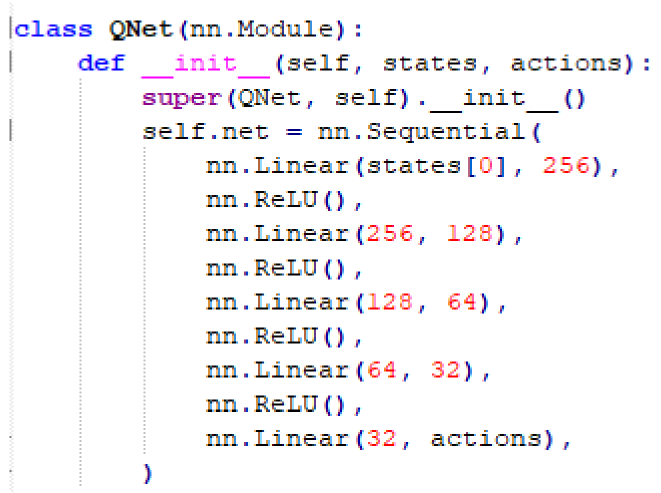
Code tested on Spyder IDE from Anaconda

Run train.py to see the code in action.

Source Code: GitHub repository

# Model

- Hyperparameters
- Neural Network Architecture



```
class QNet(nn.Module):
    def __init__(self, states, actions):
        super(QNet, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(states[0], 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, actions),
        )
```

| Hyperparameter | Value |
|---|---|
| episodes | 1000 |
| buffer_size | 100000 |
| batch_size | 64 |
| gamma | 0.99 |
| learning rate | 1e-3 |
| tau | 1e-3 |
| steps | 4 |

## Q - Learning

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left( \underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)}^{\text{temporal difference}}$$

- Used to select optimal action-selection policy.
- Decides what action to take in next place.
- Agent learns function that predicts reward of taking an action for given state. Neural Network approximates this function.

## Observation Space

- State Space – [x-position, y-position, x-velocity, y-velocity, lander angle, angular velocity, right leg grounded, left leg grounded]
- Environment Variables

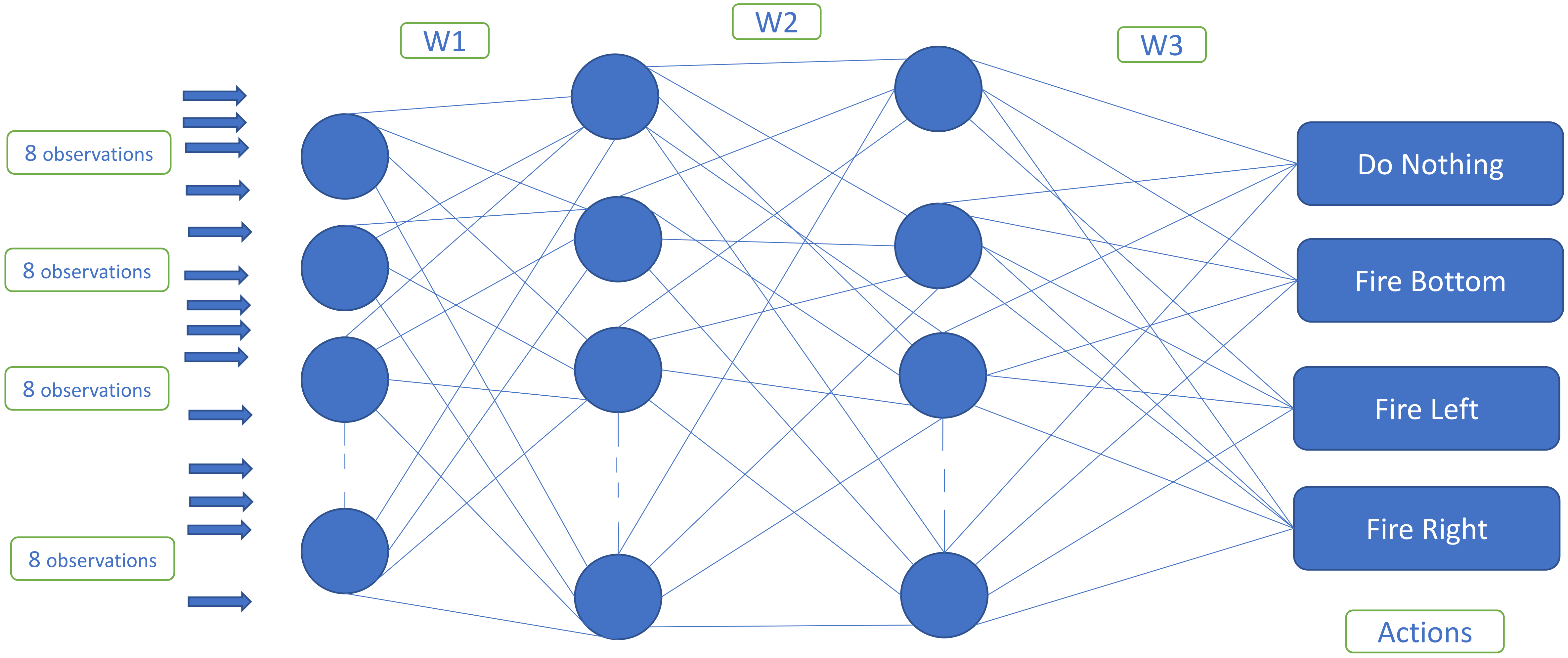
1. State – current state of the environment (8 – dim state space)

2. Action – Agent acts based on current state.

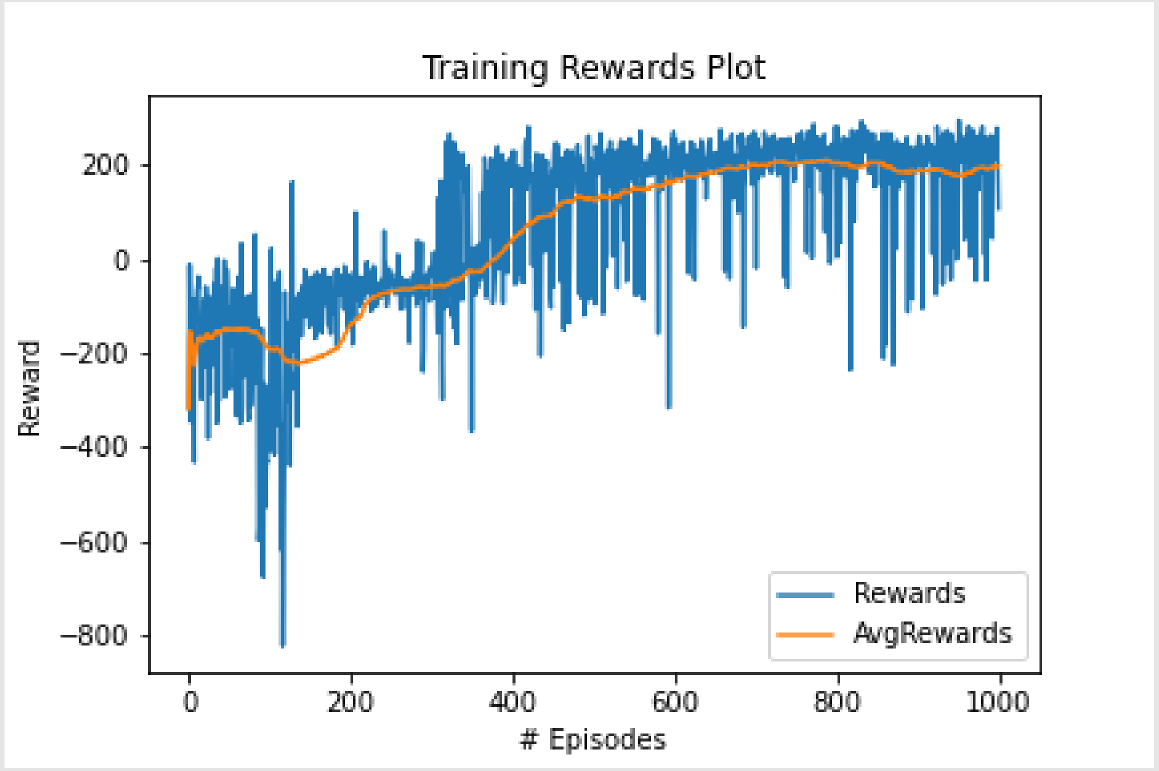
3. Reward – If lander crashes or comes to rest, the episode is considered complete and receive reward.

- Max episode count – 2000
- Stopping criteria – If average of 200 for last 100 episodes.

# Neural Network Model



# Results



Rewards obtained at each training episode with early stopping

Blue line – reward values per experiment

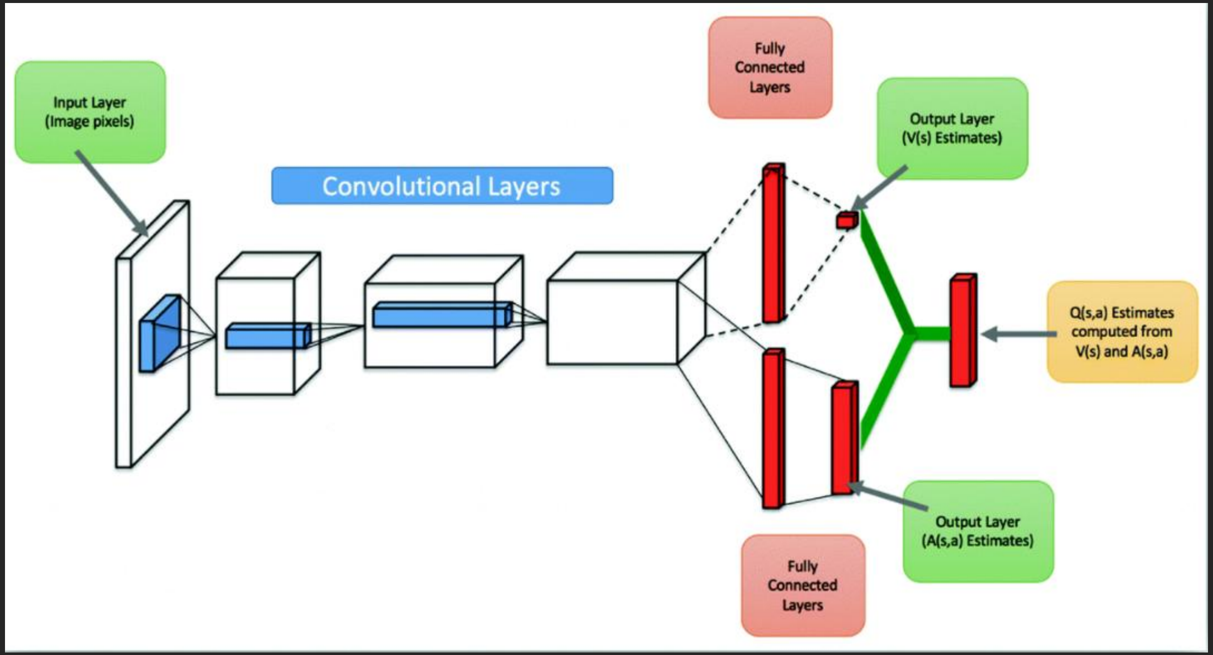
Orange line – rolling mean for last 1000 episodes.

Reward is positive after 300 episodes.

## Knowledge Gained

- Training the agent take more time.
- Improved the neural network model accordingly to solve the agent in less time.
- Used different parameter values to see how the training change over time.
- The lunar lander can solve the problem using a memory length of 100000 in less time.

# Extension – Double DQN


Input Layer (Image pixels)
Convolutional Layers
Fully Connected Layers
Output Layer (V(s) Estimates)
Q(s,a) Estimates computed from V(s) and A(s,a)
Output Layer (A(s,a) Estimates)
Fully Connected Layers

Here I extended the existing neural network using two different function approximators that are trained on different samples, and one is used for selecting the best action and other for evaluating the value of this action. since these two approximators have seen different samples, it is less that they overestimate the same action. Hence the name Double Q-Learning

## Conclusion

- The experiments conducted as part of this project solved the Lunar Lander Environment using DQN Algorithm with Soft Updates. It is observed that the weight propagation parameter τ and weight decay parameter  greedy has significant impact on the learning performance.
- One major disadvantage for environments like Lunar Lander is it ignores weight of fuel used and is a 2D simulation of a 3D environment.

Reference: https://gym.openai.com/docs/

## Future Work

- Simulator in the OpenAI gym interface for Reinforcement Learning (3D environment simulation)

Team Name: STARLANDER: The Millennium Falcon