Introduction:

The environment chosen for this task is OpenAI gym's Lunar Lander. In this environment, the agent controls a spaceship in a bidimensional world with the task of landing it safely onto the below landing pad.

Landing pad is always at coordinates (0,0).

Coordinates are the first two numbers in the state vector. Reward for moving from the top of the screen to landing pad and zero speed is about 100..140 points. If the lander moves away from the 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 the main engine is -0.3 points each frame. The agent receives 200 points if he lands inside the landing pad. Landing outside the landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt.

Action space:

Four discrete actions available:

- (0) do nothing
- (1) fire left
- (2) fire main engine
- (3) fire right

LunarLander-v2 defines **solving** as getting an average **reward of 200 over 100 consecutive trials**.

State space:

The state of the environment is represented by an eight-dimensional vector

(x, y, vx, vy, θ, ω, ldlef t, ldright).

- **(x, y)** are the lander's current coordinates.
- **(vx, vy)** are the horizontal and vertical velocities.
- θ is the angle of the lander.
- **ω** is the angular velocity.
- **(ldleft, ldright)** are two flags that are set to 1.0 if the corresponding leg of the lander makes contact with the ground.

Learning Algorithm

Project develops intelligent agents using deep reinforcement learning techniques of Deep Q-Network (DQN) and Double deep Q-Network (DDQN) to play the LunarLander game of OpenAI environment using Torch library.

DQN combines Q-learning with a deep neural network, by using neural networks as a policy and experience replay buffer and target networks.

The difference between DQN and DDQN is in the calculation of the target Q-values of the next states. In DQN, we simply take the maximum of all the Q-values over all possible actions. The chosen action is the one selected by our policy model. In DQN, the target Q-Network selects and evaluates every action resulting in an overestimation of Q value. To resolve this issue, DDQN proposes to use the Q-Network (policy model) to choose the action (Action Selection) and use the target Q-Network to evaluate the action (Action Evaluations).

Model architecture

The agent has a target and local networks having the same architecture:

- 1 fully connected layer of size 64.
- 1 fully connected layer of size 64.
- 1 output layer of size 4 (the size of the action space).

Hyperparameters

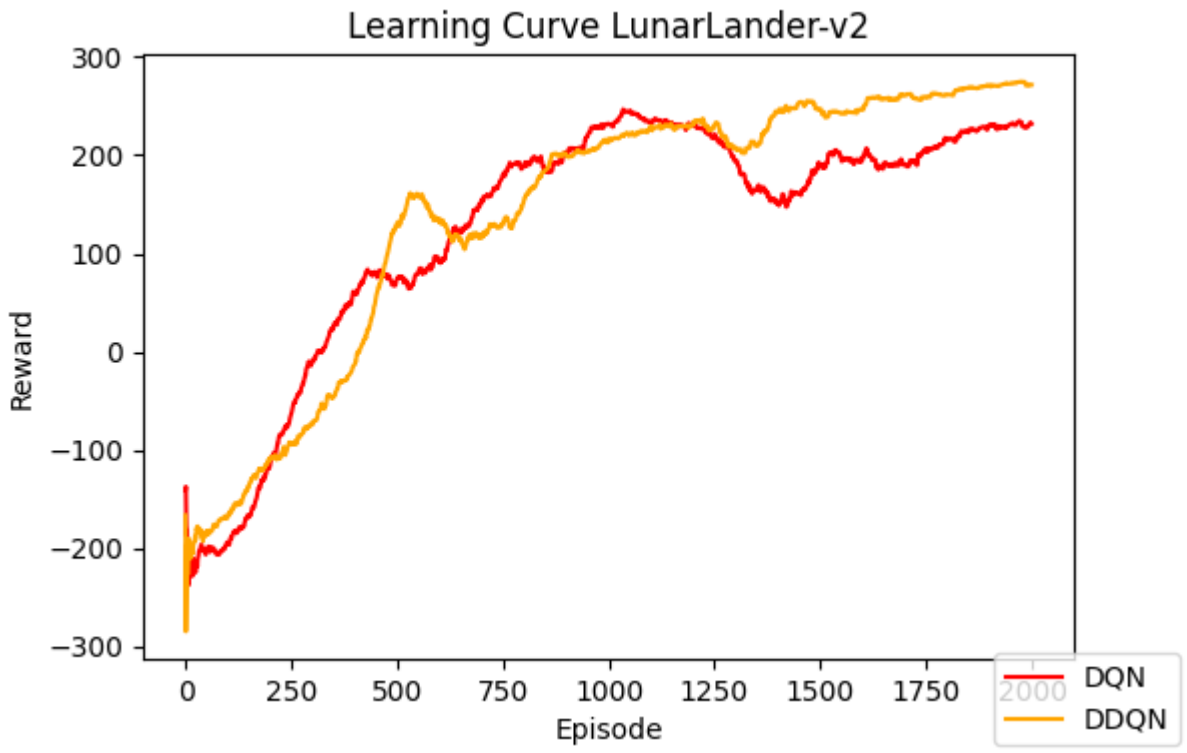
- Buffer size: the size of the experience replay buffer is 100 000;
- Batch size: the batch size of the training is 64;
- Gamma: the discount factor 0.99;
- TAU learning rate coefficient for soft update of target parameters is 0.001;
- The network is updated after every 4 time steps;
- Learning rate 0.0005;
- Number of episodes per model 2000;
- Max steps/ episode 1000.

How to train our agent

In order to train our agent we have to:

1. Initialize the agent.
2. Give the initial state of the environment to the agent.
3. For each time step, give to the agent the current state of the environment and he will return the action that he will perform.
4. Save the experience in the replay buffer.
5. Iterate until the agent reaches the score of 274.Stop the training.

Results:



Future developments:

- Tuning the hyperparameters for better results.
- Implementing Dueling DQN