

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | | Double Deep Q Network (DDQN) for the Lunar Lander (v2) | | |  | |
|  |  | | | | |  |
|  | | |  |  | | |
|  | | | RueenFiezMoghaddam |  | | |
|  | | | March 28, 2020—Final Project—3547 – Intelligent Agents & Reinforcement Learning |  | | |

#### Project Objectives

The main project objective includes implementing a Double Deep Q Network (DDQN) solution with Keras and Python to solve the OpenAI Gym’s Lunar Lander (v2) environment.

|  |
| --- |
|  |

To provide some background, this will be a lunar lander attempting to land gracefully within a defined “goal position” area defined by the environment; it will spawn from the top of your screen, and gradually work its way falling down to the bottom of your screen. The lander must be able to leverage its rockets/thrusts to navigate its way to the goal position. An episode is complete when the lander is in a resting state either via crash or graceful landing. Landing outside the goal position is allowed.

The goal position coordinates are static and the lunar lander has only four discrete actions to select:

1. Rockets/Thrust **Left**
2. Rockets/Thrust **Right**
3. Rockets/Thrust **Downward**
4. Do **Nothing** / **No** Action

Solved for an episode is defined as reaching a reward of +200. We track the Agent’s average rewards across a span of 100 episodes. Our solution uses 500 episodes for testing purposes where an average reward of 50-75 should serve as an acceptable solved baseline. This is our main objective for the DDQN project.

**Rewards scheme details are as follows:**

* Lander’s Leg ground contact: +10 reward
* Firing Rockets/Thrust Downward: -0.3 reward
* Crash landing is an additional: -100 reward
* Graceful landing is an additional: +100 reward
* Moving away from the goal position will reduce reward
* Moving closer to the goal position will increase reward

#### Design Detail

Aim was to use a DDQN to eliminate the maximization bias which is found when using a DQN and evaluate how this performs in the Lunar Lander (v2) environment while fine-tuning some hyperparameters.

By leveraging a target network which remains unchanged for t time which has the Q function (action value) we can avoid the overestimating which occurs in Q learning for the action-value estimates and the value function estimates.

We are using two neural networks to approximate our Q-values:

* **Q\_Eval (Q primary)**
  + Instantiated in the AGENT\_DDQN class
  + Trained with state / Q\_Target
  + Used for action selection
    - When predicting Q values used for action selection, it uses the new\_state
  + When selecting an action, it predicts using the current state (including initial state)
* **Q\_Target (Q‘ target)**
  + Instantiated in the AGENT\_DDQN class
  + Predicts Q values using the new\_state
  + There is no fitting for this DQN, it has its weights updated from the Q\_Eval DQN
    - In code, the method network\_update() based on the variable *FREQ\_UPDATE\_TARGET\_NETWORK=100*
  + Used for action evaluation

Using an epsilon-greedy strategy for exploit vs explore, and have been experimenting with resetting the epsilon value whenever the epsilon reaches its minimum point (hyperparameter), however the latter resetting approach did not yield optimal results for our lunar lander and have later chosen to turn off epsilon reset (see Agent Performance section below for further details).

To tackle the issue of “catastrophic forgetting” we implement a class called MEM\_BUFFER. Our MEM\_BUFFER class allows the DDQN Agent to sample transitions across different episodes.

We sample random experiences using the sample\_buffer method in our MEM\_BUFFER class.

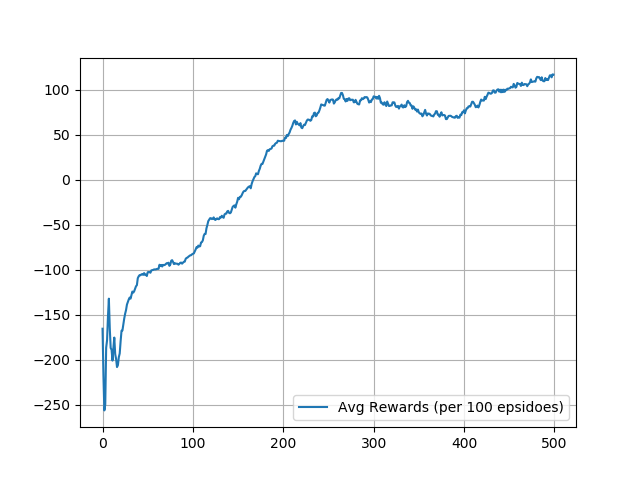
#### Agent Performance

Evaluation of the DDQN Agent’s performance via the rewards tracking during each episode to highlight the “average rewards” across 100 episodes. For this project, the DDQN Agent always played 500 total episodes (due to time and hardware limitations). Based on performances displayed below, assume additional episodes would result in increased average rewards however this would need to be proven via additional testing and probably some hyperparameter tuning.

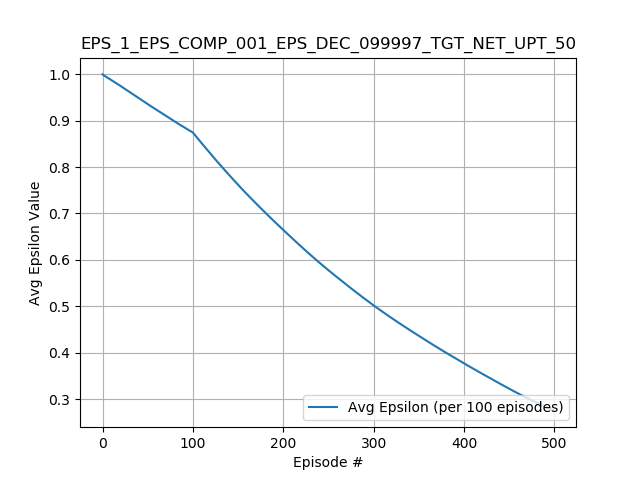
**Baseline / Initial**

**| EPISODE= 499 | SCORE= -63.80 | AVERAGE\_SCORE= 108.68184935495806 | Averaged over: 100 episodes |**

***Hyperparameters:*** *NO\_OF\_EPISODES=500, FREQ\_UPDATE\_TARGET\_NETWORK=100, EPSILON\_DECAY\_RATE=0.98,EPSILON\_COMPLETE=0.01, MEMORY\_SZ=1000000, DENSE\_1\_SZ=64, DENSE\_2\_SZ=64,#DROPOUT\_SIZE=0.20, ALPHA=0.0008, GAMMA=0.99, POSSIBLE\_ACTIONS=4, EPSILON=1.0, BATCH\_SZ=64, INPUT\_D=8*



##### When using a *very* slow decay rate for our epsilon-greedy strategy (no epsilon reset), there was no avg rewards improvement over the original baseline above. This showed that a lengthy epsilon did not translate to improved scores for this env.



Finally, one of the better DDQN\_Agent performances below, along with the hyperparameters used:

**Best Results (beats our goal/objective of 50-75 avg rewards (per 100 episodes) across 500 episodes):**

**| EPISODE= 499 | SCORE= 249.13 | AVERAGE\_SCORE= 116.65603067898283 | Averaged over: 100 episodes |**

Machine generated alternative text:
100 
—50 
—100 
—150 
—200 
—250 
100 
Avg Rewards (per 100 epsidoes) 
200 
300 
400 
500 

***Hyperparameters:****NO\_OF\_EPISODES=500,FREQ\_UPDATE\_TARGET\_NETWORK=100,EPSILON\_DECAY\_RATE=0.98,EPSILON\_COMPLETE=0.01,MEMORY\_SZ=1000000,DENSE\_1\_SZ=64,DENSE\_2\_SZ=64,ALPHA=0.0008,GAMMA=0.99,POSSIBLE\_ACTIONS=4,EPSILON=1.0*

##### Video Demonstration & Github Link (500 Episodes Compilation)

This will demonstrate the lunar lander agent’s early, mid, and later learning to reach the goal position in the environment over 500 episodes. Also github link to see more info on the solution.

* **Youtube** Link: <https://youtu.be/b_X4uTWkkRY>
* **Github** Link: <https://github.com/rfiez/DDQN_LL_UOFT_FP>

Thank you