# Reinforcement Learning - Bonus Part

**Kiran Prabhakar - kprabhak - 50287403**

## Abstract

The purpose of the bonus part in the current project is to try out the reinforcement learning task in the environment chosen from the OpanAI's Gym library.

## 1 Introduction

The current bonus part of the project uses the Cartpole environment of the Open AI's Gym library. The implementation is done by exploiting the DQN mechanism provided in the stable baseline.

## 2 CartPole environment

### 2.1 Time-steps:

Time steps is the number of steps or actions that the cart pole takes in the environment. In the current cart pole environment, the agent has two possibilities, moving left or right so balance the pole.

### 2.2 Reward Criteria:

- The reward +1 is provided for every time stamp the agent remains upright.
- The episode ends when the pole is more than $15^o$ from the vertical or the cart moves more than 2.4 units.

### 2.3 Success criteria:

The cart pole is said to be solved when the mean average reward is of 195 over 100 consecutive episodes.

### 2.4 Stable Baseline

Stable base line is a well known library that gives a number of pre-defined and well known algorithms for performing the reinforcement learning task. In the current project the stable baselines implementation of DQN is used to perform the reinforcement learning task.

### 2.5 Methods Leveraged:

#### 2.5.1 From Gym:

The functionality of methods provided by Gym are same as the methods provided in the current project.

- **Make**: The method accepts a parameter specifying about the type of environment to create. In the current project **CartPole-v1** is passed to retrieve cartPole environment.
- **step**: Allows the agent to take an action. The method takes agent action as input.

Preprint. Work in progress.

- **Reset**: Gives a new environment and is called for every episode.
- **Render**: Renders the environment for the agent to act upon.

### 2.6 From Stable baseline:

- **DQN**: The DQN constructor accepts different parameters for creating a DQN model. The most important parameters that are tweaked in the current project are policy, batch_size, exploration_fraction, exploration_final_eps.
- **learn**: The method is called to get the trained model. The important parameter is the number of time steps required, the model is allowed to train upon.
- **predict**: Returns a tuple which is the models action and predicted next state.
- **Policy**: The library provides four different policy objects to implement DQN, it is passed as parameter for the DQN constructor. The avalible policies are MlpPolicy, LnMlpPolicy, CnnPolicy, LnCnnPolicy. In the current project MlpPolicy and CnnPolicy are tried out.

The further details of the usage of the methods are provided as comments in the code.

### 2.7 DQN Performance on Environment:

The performance of the DQN is measured by tweaking various parameters accepted by the methods from the stable baseline.

The below table represents the parameter's that are experimented on stable baseline environment for 100, 128 as batch size iterations and 0.3 as exploration fraction.

| Policy | total_timesteps | exploration_fraction | Reward |
|---|---|---|---|
| MlpPolicy | 20000 | 0.3 | 140 |
| MlpPolicy | 50000 | 0.3 | 170 |
| MlpPolicy | 70000 | 0.3 | 243 |

Thus from the above table it is evident that by properly choosing the number of time-steps, the agent could learn the environment properly.

### 2.8 Execution Logs:

From the recent execution with the 70000 time-steps, the output of the program is as below.

#### 2.8.1 100 episodes:

| | |
|---|---|
| % time spent exploring | 85 |
| episodes | 100 |
| mean 100 episode reward | 22.6 |
| steps | 2238 |

#### 2.8.2 200 episodes:

| | |
|---|---|
| % time spent exploring | 62 |
| episodes | 200 |
| mean 100 episode reward | 34.8 |
| steps | 5721 |

#### 2.8.3 Final 100 episodes:

| | |
|---|---|
| % time spent exploring | 2 |
| episodes | 1200 |
| mean 100 episode reward | 243 |
| steps | 70000 |

### 2.9 Observations:

- From the above execution logs it is evident that the agent initially spends a large percentage (85 in the current case) of time exploring the environment but with a penalty of minimum reward.
- However as the agent continues to explore the environment, it doesn't find much to learn any further. So, it spends a minimum time exploring the environment (2% of the time)
- **predict**: In the final episode the agent exploits a lot from its learning and then attain maximum mean episode reward of (243) for the final episode.

Thus the in built DQN function provided by stable baseline, helps the agent to explore and exploit in the environment. The implementation details for the current DQN function is provided in the code with comments.

## References


[1] https://towardsdatascience.com

[2] https://medium.com/data-science-group-iitr

[3] https://chemicalstatistician.wordpress.com