



◀ [Return to "Deep Reinforcement Learning Nanodegree" in the classroom](#)

[DISCUSS ON STUDENT HUB](#)

# Collaboration and Competition

## REVIEW

## CODE REVIEW

## HISTORY

### Meets Specifications

Dear Udacian, the project is very well implemented and meets the specifications! Congratulations on successfully completing the project.

As a next step, please go through the resource: [human-like robot hand trained to manipulate physical objects with unprecedented dexterity](#).

All the best! 😊

### Training Code

The repository includes functional, well-documented, and organized code for training the agent.

The repository contains jupyter notebook, code files, readme, and project report. The code is functional. Great job solving the environment using Multi Agent Deep Deterministic Policy Gradients.

### Suggestions

- Using batch normalization helped to achieve better results quite significantly.
- I request you to try using batch normalization on the input to both the actor and critic networks.

The code is written in PyTorch and Python 3.

The code is written in pytorch and python3.

- A good read: [PyTorch vs TensorFlow—spotting the difference](#)

The submission includes the saved model weights of the successful agent.

Thanks for including the saved model weights of the successful agent.

## README

The GitHub submission includes a `README.md` file in the root of the repository.

The README describes the the project environment details (i.e., the state and action spaces, and when the environment is considered solved).

Awesome work providing the project environment details in the README.

State space, action space, reward function and when the environment is considered solved is specified very informatively.

The README has instructions for installing dependencies or downloading needed files.

Proper instructions have been specified in the README to download the necessary files.

The README describes how to run the code in the repository, to train the agent. For additional resources on creating READMEs or using Markdown, see [here](#) and [here](#).

## Report

The submission includes a file in the root of the GitHub repository (one of `Report.md`, `Report.ipynb`, or `Report.pdf`) that provides a description of the implementation.

Report.md has been included in the root of the github repository.

The report clearly describes the learning algorithm, along with the chosen hyperparameters. It also describes the model architectures for any neural networks.

The report is rather informative providing an insight on every aspect of the project which includes Implementation, model architectures, hyperparameters, rewards, future works.

- Good choice to implement the MADDPG algorithm.
- It is found to work very well with continuous action space and multiple agents.
- Good Implementation of the Actor and Critic networks.
- Good decision to use replay buffer to store and recall experience tuples.
- Good job using the target networks for Actor and Critic, as suggested in the original paper.
- Good choice to use tau to update the target network.

A plot of rewards per episode is included to illustrate that the agents get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).

The submission reports the number of episodes needed to solve the environment.

## Awesome

- The agent seems to perform very well!
- The agent is able to achieve a reward of 0.5+ over last 100 episodes in just 1278 episodes!
- The submission discusses the rewards plot obtained clearly.

The submission has concrete future ideas for improving the agent's performance.

Great job providing the ideas to experiment more in future with the project!

You should try implementing Prioritized Experience Replay also. It helps to improve the performance and significantly reduces the training time. This should also help to stabilize the performance to some extent. A fast implementation of Prioritized Experience Replay is possible using a special data structure called Sum Tree. I found a [good implementation here](#).

Also, I request you to check the following posts to get familiar with more reinforcement learning algorithms.

- [Asynchronous Actor-Critic Agents \(A3C\)](#)
- [Trust Region Policy Optimization \(TRPO\) and Proximal Policy Optimization \(PPO\)](#)

Here is an [implementation](#) of PPO on tennis environment. The training was slow but the final average score achieved was almost 1.25 (with some fluctuation). You should surely try PPO in future.

 [DOWNLOAD PROJECT](#)

RETURN TO PATH

---