UDACITY

‹ Return to "Deep Reinforcement Learning Nanodegree" in the classroom

DISCUSS ON STUDENT HUB

Continuous Control

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Great job getting familiar with the Deep Deterministic Policy Gradients and successfully completing the project. This project was a lot tougher over the last one. Great job! The next project is even more exciting and awaits you. All the best! 🌟
The recent achievement of the Open AI group to play Dota 2 using Reinforcement Learning is a must read. 😄

Training Code

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

Awesome work implementing a reinforcement learning agent to solve the "reacher" environment.

- Very good decision to implement the Deep Deterministic Policy Gradients algorithm, a very effective reinforcement learning algorithm.

PROS OF THE IMPLEMENTATION

- Implementing the DDPG algorithm is a good choice as it is found to work very well with continuous action space and state space.
- Correct Implementation of the Actor and Critic networks.
- Good use of replay memory to store and recall experience tuples.
- Using the target networks for Actor and Critic networks is a good choice.
- Good choice to update the target network using soft updates and using tau for it.

The code is written in PyTorch and Python 3.

Awesome work completing the project using PyTorch.

- Tensorflow and PyTorch are the two most competing choices for Deep Learning Applications. It would be good to check Tensorflow or PyTorch : The Force is Strong with which One?

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

The saved model weights of the successful agent are there.

README

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

Thank you for providing the README file for the project.

- A README file helps tell other people why your project is useful, what they can do with your project, and how they can use it.
- It is an industry standard practice and helps to make the repository look professional.

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 including the state and action spaces, the reward function and when the agent is considered solved. The description is very informative.

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

Awesome work providing the instructions for installing the dependencies and downloading the environment.

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.

Awesome work including the instructions to run the code.

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.

Awesome work writing the report of the project with the complete description of the implementation. All the sections are detailed.

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

Great job covering the learning algorithm, hyperparameters, details of the neural networks, plot of rewards and the ideas for future work in the report.

A plot of rewards per episode is included to illustrate that either:

- ***[version 1]*** **the agent receives an average reward (over 100 episodes) of at least +30, or**
- ***[version 2]*** **the agent is able to receive an average reward (over 100 episodes, and over all 20 agents) of at least +30.**

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

Performance of the agent is awesome. Score of 30.07, averaged over last 100 episodes, is achieved in just 407 episodes.

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

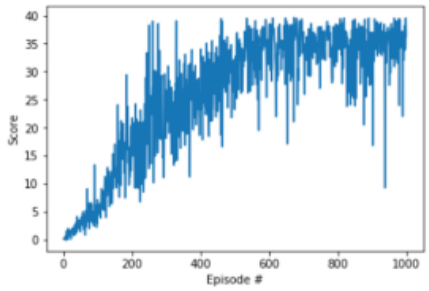
Thank you for providing the details of the experimentation you intend to do in the future!

An effective way to improve the performance of DDPG is by using Prioritized Experience Replay. You should check this github repo for a fast implementation of Prioritized Experience Replay using a special data structure Sum Tree.

Below is a comparison of DDPG with random sampling vs DDPG with PER for the Reacher environment. It's quite evident how episode variation decreased and performance improved.

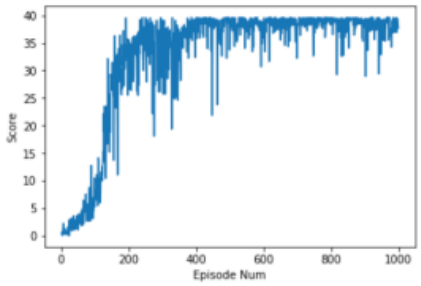
Random Sampling:

```
Episode 100     Average Score: 2.90     Score: 7.17
Episode 200     Average Score: 11.84    Score: 21.75
Episode 300     Average Score: 19.88    Score: 18.15
Episode 400     Average Score: 24.14    Score: 23.15
Episode 500     Average Score: 28.95    Score: 23.24
Episode 600     Average Score: 33.21    Score: 38.19
Episode 700     Average Score: 34.68    Score: 33.98
Episode 800     Average Score: 34.64    Score: 35.97
Episode 900     Average Score: 34.15    Score: 37.01
Episode 1000    Average Score: 35.13    Score: 36.33
```



PER:

```
Episode 100     Average Score: 2.96     Score: 6.03
Episode 200     Average Score: 13.11    Score: 35.57
Episode 300     Average Score: 20.10    Score: 38.79
Episode 400     Average Score: 30.82    Score: 39.54
Episode 500     Average Score: 35.71    Score: 33.71
Episode 600     Average Score: 37.06    Score: 35.91
Episode 700     Average Score: 38.11    Score: 37.30
Episode 800     Average Score: 38.40    Score: 39.57
Episode 900     Average Score: 38.48    Score: 37.85
Episode 1000    Average Score: 38.40    Score: 37.68
```



Following posts give an insight into some other reinforcement learning algorithms that can be used to solve the environment.

- Proximal Policy Optimization by Open AI
- Introduction to Various Reinforcement Learning Algorithms. Part II (TRPO, PPO)

DOWNLOAD PROJECT

RETURN TO PATH