Udacity-Review for the RL-Project „Navigation“ 22.04.2010

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 for the next projects! 😄

Training Code

✓ **The repository (or zip file) 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.
The implemented Deep Q Network and Double Deep Q Network look good!

You should definitely check the following resource to progress with the learning further:

- Speeding up DQN on PyTorch: how to solve Pong in 30 minutes
- Advanced DQNs: Playing Pac-man with Deep Reinforcement Learning by mapping pixel images to Q values

✓ **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 (or zip file) 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 install the dependencies and 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 or zip file (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 implementation of the agent for Deep Q Network and Double Deep Q Network.
- Correct decoupling of the parameters being updated from the ones that are using a target network to produce target values.
- Perfect implementation of The Epsilon-greedy action selection to encourage exploratory behavior in the agent.
- Good use of tau parameter to perform soft-update. It helps to prevent variance into the process due to individual batches.
- Good use of the replay memory to store and recall experience tuples.
- The implementation is easy to debug and easily extensible.

✓ **A plot of rewards per episode is included to illustrate that the agent is able to receive an average reward (over 100 episodes) of at least +13. 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 13+ over last 100 episodes in just
  - 464 episodes for DQN!
  - 356 episodes for DDQN!
- 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!

I would like to point you to the following resources:

- Rainbow: Combining Improvements in Deep Reinforcement Learning
- Conquering OpenAI Retro Contest 2: Demystifying Rainbow Baseline

As pointed in the report, you should try implementing Prioritized Experience Replay also. It helps to improve the performance and significantly reduces the training time. A fast implementation of Prioritized Experience Replay is possible using a special data structure called a Sum Tree. I found a good implementation here.

And, you should definitely try applying these algorithms by taking raw screen pixels as input also. In case you get stuck, definitely check this github repository.