Meets Specifications

Dear Udacian,

Great job getting acquainted with the Deep Q Network and Double Deep Q Network algorithms and successfully implementing them to complete the project. 😄

I would suggest you to go through Deep Reinforcement Learning for Self Driving Car by MIT. You'd get to know more about reinforcement learning algorithms in broader and real-world perspective and, more importantly, how to apply these techniques to real-world problems.

All the best for future endeavors. ✨

Training Code

The repository (or zip file) includes functional, well-documented, and organized code for training the agent.

Awesome

- 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.
- Good implementation of The Epsilon-greedy action selection to encourage exploration.
- Good use of tau parameter to perform soft-update to prevent variance in the process caused by individual batches.
- Good use of the replay memory to store and recall experience tuples.

The code is written in PyTorch and Python 3.

Awesome

The code is written in PyTorch and Python 3.

Lately, PyTorch and TensorFlow happen to be most extensively used frameworks in deep learning. It would be good to get some insight by comparing them, please see the following resources:

- Sebastian Thrun on TensorFlow
- PyTorch vs TensorFlow — spotting the difference
- Tensorflow or PyTorch : The Force is Strong with which One?

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

Awesome

- Saved model weights of the successful agent have been submitted.

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

- Readme file describes the project environment details properly.
- Information about the state and action spaces, and when the environment is considered solved has been provided.

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

Awesome

- Great work providing the all the necessary installation instructions.

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

- Great work providing necessary instructions to run the code.

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.

Awesome

- Report for the project with all the details of the implementation has been provided in the submission.

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

Awesome

Great work providing the details of the implemented agent. Details of the learning algorithm used, hyper-parameters, and architectural information of the deep learning model have been provided.

- Good decision to choose DQN and Double DQN algorithms for the discrete action space problem.
- Good work including model architecture in the report.
- Hyperparameters you have used seem to be good.

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

- Discussion for the rewards is provided in the report.
- The rewards plot seems to be good and average score of +13 is achieved.

Reinforcement learning algorithms are really hard to make work.
But it is substantial to put efforts in reinforcement learning as it is close to Artificial General Intelligence.
This article is a must read: Deep Reinforcement Learning Doesn't Work Yet

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

Awesome

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

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.