Navigation

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Kudos ! I think you've done a good job of implementing the Deep Q Network to teach the agent how to collect bananas! It's very clear that you have a good understanding of the basics.

Further Reading: Teaching an agent in simulation is nice, but how do you transfer these techniques in real world problem, say a real robotics arms? I would highly recommend this read, Learning Dexterity by OpenAI, where they trained a human-like robot hand to manipulate physical objects with unprecedented dexterity.

And Congratulations 🎉 once again, for successfully this project.

Training Code

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

You included well-organized zip file with a README.md, a jupyter notebook, and the code files.

The code is written in PyTorch and Python 3.

The submission includes 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).

The README has instructions for installing dependencies or downloading needed 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.

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

Overall you did a great job implementing layers for the model. Let me illustrate the pros of the architecture you chose.

Pros

- Successfully implemented the Agent for DQN network.
- Decoupled the parameters being updated from the ones that are producing target values by using a target network.
- The Epsilon-greedy action selection is perfectly implemented to encourage exploratory behavior in our agent.
- Used tau parameter to perform soft-update, preventing variance into the process due to individual batches.
- Correctly used replay memory to store and recall experience tuples.
- Your implementation is highly modular, making it easy to debug and easily extensible.

Further Resource: Keras has a neat API for visualizing the architecture, which is very helpful while debugging your network. `pytorch-summary` is a similar project in PyTorch.

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

- Your agent is a great learner, as evident from reward plot.
- The agent is able to achieve the desired +13 reward over past 100 episodes!
- The submission correctly reports the number of episodes required to solve the task. 👍

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

Woah, these suggestions are great! ✌️
But do you know what is the current State-Of-The-Art (SOTA) RL algorithm for discreet action space? Well, when you combine of all these (and more) together, you get Rainbow. This is a great post explaining how to combine all of these improvements together.

DOWNLOAD PROJECT