UDACITY

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

DISCUSS ON STUDENT HUB

Navigation

REVIEW

CODE REVIEW

HISTORY

Requires Changes

1 SPECIFICATION REQUIRES CHANGES

Great job getting familiar with the Deep Q Network and implementing it to successfully train the agent. Great job. 😄

There is one point in the review which does not meet specifications as per the project rubric. I have provided information for it in the review.
You have trained the agent successfully, which is really appreciated, but README also needs to be completed according to the rubric. Please see the points in the review and make the required changes. I hope it's not an issue, thank you for the cooperation.

Meanwhile, the recent achievement of the Open AI group to play Dota 2 using Reinforcement Learning is a must read.

Training Code

The repository (or zip file) includes functional, well-documented, and organized code for training the agent.
Awesome work implementing a reinforcement learning agent to collect the "yellow bananas in a large square world".

- Very good decision to implement the DQN algorithm, a very effective reinforcement learning algorithm.

PROS OF THE IMPLEMENTATION

- Good implementation of Deep Q Network.
- Decoupling of the parameters being updated from the ones that are using a target network to produce target values has been done correctly.
- The Epsilon-greedy action selection to encourage exploratory behavior in the agent been perfectly implemented.
- Soft update helps to prevent variance into the process due to individual batches. Good use of tau parameter to perform soft-update.
- Good use of the replay memory to store and recall experience tuples.

As a next step, I highly suggest you to use the same algorithm to train an agent on an environment just by taking the screen pixels as the input. This would be more complicated and would require to use a more powerful convolutional neural network.
As an example, you can check Using Keras and Deep Q-Network to Play FlappyBird

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 (or zip file) 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.

Required

- Please provide, in the README, the instructions to execute the code in the repository.

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 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 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.

Performance of the agent is awesome. Score of 13.02, averaged over last 100 episodes, is achieved in just 470 episodes in the best case.

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!

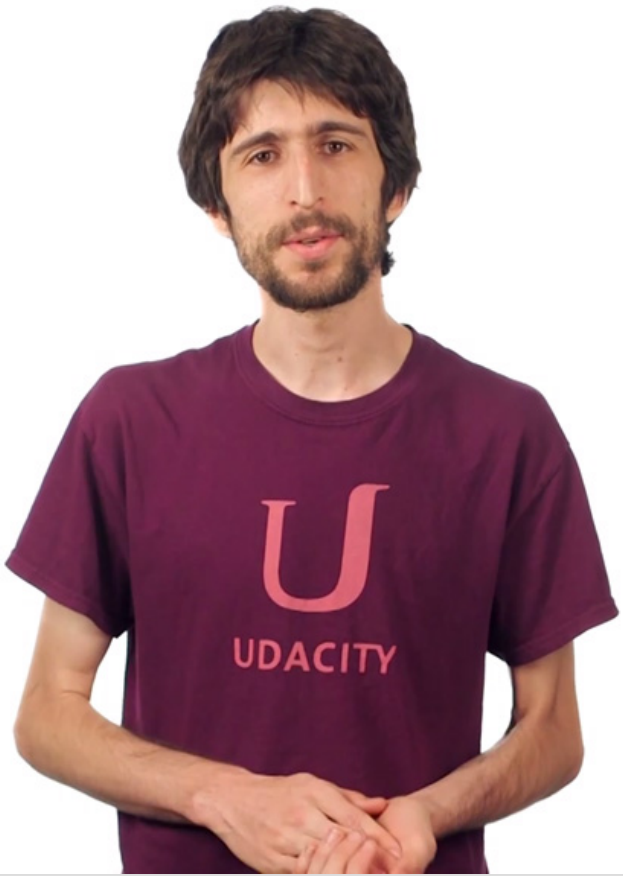
It would be very useful to check Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed Q-targets

One effective way to improve the performance 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.

And yes, you should definitely try the agent by using raw screen pixels as input. You can check here for the raw pixels implementation for reference.

RESUBMIT

DOWNLOAD PROJECT



Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH