UDACITY

Return to Classroom

Navigation

REVIEW

CODE REVIEW

HISTORY

Requires Changes

2 specifications require changes

Dear Udacian,

Great job getting acquainted with the Deep Q Network algorithm and successfully implementing it to solve the Navigation environment. The implementation is pretty good and the agent's performance is impressive as it achieves an average score of +16. The architecture used for the Q network is decent in size with two hidden layers of size 64 units each. Good work using the `relu` activation function. The report is quite informative and covers the different important aspects of the implementation. 😄

There are some rubric point which do not meet the specifications. They are related to the following:

- Providing detailed instructions to download the project environment in the README.
- Providing detailed instructions to install the dependencies in the README.
- Providing some additional information about the hyperparameters used in the project.

I have given explanation for them. Please go through the review, make the specified changes and I am sure the next submission would meet all the specifications. Looking forward to the next submission eagerly. All the best!

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.

how to apply these techniques to real-world problems.

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.
- 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.
- The `checkpoint.pth` file is present in the submission.

README

The GitHub (or zip file) submission includes a `README.md` file in the root of the repository.

Awesome

- Great work documenting the project details and submitting the README file.

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 in the `Introduction` section.
- 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.

Required

The following information is provided in the `Getting Started` section of the README.

```
1. Clone this repository.
2. Install Unity on your system. Please check out your python environment with pr
oper dependencies before beginning with this project.
```

The instructions should be more detailed. Please consider the following points:

- It is requested to add the instructions to download the project environment.
- It is also requested to provide the instructions to download the dependencies, to set up the python environment.
- You may refer to the DRLND GitHub repository. These instructions are necessary to complete the project.

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 in the `Instructions` section.
- All the cells in `Navigation.ipynb` file should be executed to train the agent.

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 algorithm for the discrete action space problem.
- Good work including model architecture in the report.

  Fully connected layer 1 with ReLU (input: 37 units (state_size), output: 64 units)

  Fully connected layer 2 with ReLU (input: 64 units, output 64 units)

  Fully connected layer 3 (input: 64 units, output: 4 units (action_size))

- Hyperparameters you have used seem to be good.

  Maximum steps per episode: 1000

  Epsilon-greedy decay: 0.995 (eps_start: 1.0, eps_end: 0.01)

Required

- Thank you for providing the hyperparameter values in the report.

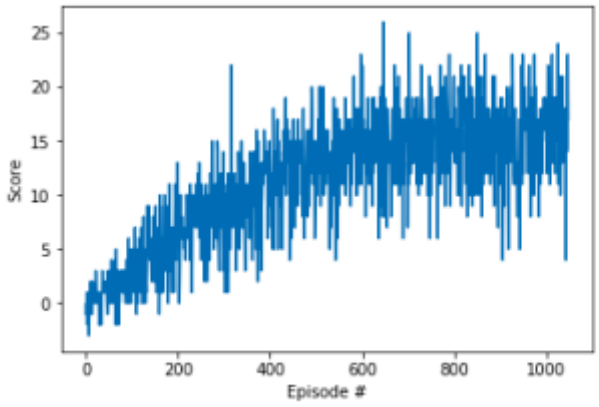
- However, it is requested to also provide the values for the following hyperparameters as well:
  `BUFFER_SIZE, BATCH_SIZE, GAMMA, TAU, LR, UPDATE_EVERY`

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 in 600 episodes.
- Furthermore, the agent achieves the maximum average score of +16!

```
Episode 100     Average Score: 1.00
Episode 200     Average Score: 4.42
Episode 300     Average Score: 7.87
Episode 400     Average Score: 9.45
Episode 500     Average Score: 12.42
Episode 600     Average Score: 13.88
Episode 700     Average Score: 14.67
Episode 800     Average Score: 15.75
Episode 900     Average Score: 15.23
Episode 1000    Average Score: 15.13
Episode 1048    Average Score: 16.03
Environment solved in 948 episodes!     Average Score: 16.03
```



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

Impressive work listing the following concrete future ideas for improving the agent's performance:

- Double DQN
- Dueling DQN
- Using CNN to solve the pixel based environment
- Prioritized Experience Replay
- Hyperparameter optimization

- Using parameter noise

Suggestions

It is requested to check the following resources to get familiar with the Rainbow algorithm:

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

As specified in the report, a possible improvement is Prioritized Experience Replay (PER) which helps to improve the performance. PER should also help to significantly reduce the training time also. Using Sum Tree, a special data structure, a fast implementation of PER is possible. It is requested to check this implementation.

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH