Introduction

In general, the problem with reinforcement learning (RL) course is, that I was not able to run basically any code given in the RL course repository locally without bigger or smaller changes. This became much more severe with the Navigation task, where I could not make the environment run locally on my machine. The help from Udacity forum was not helpful (if any, within a reasonable time frame).

Creating a code that I am not able to run on my own machine in my own environment violates reproducibility and as a consequence is not useful at all. So in the end I decided to do the project the following way to have both functional code in my environment and fulfill the requirements given by Udacity for passing the taks:

1. I developed the code using AI Gym environments and python environment that can be reproduced by following the Installation and Set Up sections in the readme file. The main code is in *notebooks/final/NAVIGATION_*.py*. The code and logic are explained in more details in this document.
2. Afterwards, I copied all the necessary code hard-way into Udacity's Colab environment into the Navigation.ipynb notebook and updated it to be functional within the environment. The code is not imported there but only copied. The reason is that I wanted to be fully transparent and reproducible, so no external unknown code imports. After the performance matched the criteria, the notebook was downloaded as a **.ipynb* file with several outputs in **.html* format to show the result. This, altogether with the model parameters for respective runs can be found in */notebooks/colab* folder. **THIS CODE IS NOT WORKING WITHIN THE ENVIRONMENT SPECIFIED IN THIS NOTEBOOK FOR THE REASON DESCRIBED ABOVE.** But I hope that this way everything is as much transparent as possible.

Components of Agent

My personal goal for this take was to check improvements of the agent from a simple Q network closer to a full RAINBOW agent. The primary inspiration was taken from the tasks/code presented during the lectures in the Udacity course. Then, I created my own implementation of the core functionality, the way I am used to work with the data so that the solution can be easily used within my framework. In addition, I searched for resources and advanced solutions, and in the end, I incorporated some of them into the base I created to get much more effective solution. I cite the source in these cases, and if I read the Udacity requirements correctly, it is allowed to do it this way in case of proper citation.

The reason for this approach is not only to come up with my own implementation but to test the ability to incorporate advanced solutions that are already available. This way simulates a more realistic set-up because these days is very important to be fast and effective in getting to the functional solution. I sincerely hope that this is acceptable.

Environments

The Cart Pool and Lunar Lander environments were used for developing the final agents. The overwiew of the agents together with their methods needed for the development can be found in the notebook */notebooks/final/NAVIGATION_00_environments.py* or in its frozen version in */docs/NAVIGATION_00_environments.html*.

Replay Buffers

Two replay buffers were implemented.

First, a simple numpy-based replay buffer. The class `ReplayBuffer` is located in */src/replay_buffer.py,* and the usage documentation is in the notebook */notebooks/final/NAVIGATION_01_replay_buffer.py* or in its frozen version in */docs/NAVIGATION_01_replay_buffer.html*.

Next, the goal was to implement a prioritized experience replay buffer. A basic search was done, and the following interesting resources were found:

- Article one.
- Article two.

The outcome of the search was that a naïve implementation would not be very effective with respect to computational complexity. A binary tree approach needs to be used to reduce the complexity of the memory. In the end, a segment tree from this source was used. The code is located in */src/external/ segment_tree.py,* together with the link to its source.

Finally, the `PrioritizedReplayBuffer` class located in /src/data/replay_buffer.py was created. The inspiration was taken from this source, but the code was adopted to the introductory `ReplayBuffer` class, which is a parent class.

Double Q Learning

A double Q learning agent was used.

Neural Networks

The neural network part was taken easily, mainly from the Udacity course tasks, just replaced with functional API. Moreover, the duelling network was created too.

The networks are located in */src/models/torch_networks.py*:

- QNetwork. With two hidden dense layers with relu activation. The size of the hidden layers is 64.
- DuelingQNetwork. The feature layer was taken the same as in the previous network. The advantage and value layers have just one more linear layer.

In addition, the above networks with dropout layers were tested, but the performance was not good. The motivation for that was the signs of overfitting during the longer training sessions. In the end, the early stopping slightly over the requested accuracy was used to overcome overtraining. But for sure, this is a topic for more investigation, why regularisation works that poorly.

Finale Agents

More agent's configurations were used and tested, but double Q learning agens was used as a baseline.

- Double Q agent,
- Replay buffer vs. prioritized replay bugger,
- Normal neural network vs. dueling neural network,
- Not regularised networks vs. regularised networks.

The variant Double Q network + prioritized replay bugger + dueling neural network was used for the final agent.

The code for training the agents can be found in:

- */notebooks/final/ NAVIGATION_02_double_q_agent_training.py*

- */notebook/final/NAVIGATION_02_double_q_agent_with_PER_training.py*

Sample runs are available in:

- */docs/ NAVIGATION_02_double_q_agent_training.html.*
- */docs/NAVIGATION_02_double_q_agent_with_PER_training.html.*

Results

Testing Phase

First of all, the variants of the agent were tested. The results can be found in */docs/Navigation*.

Udacity Task

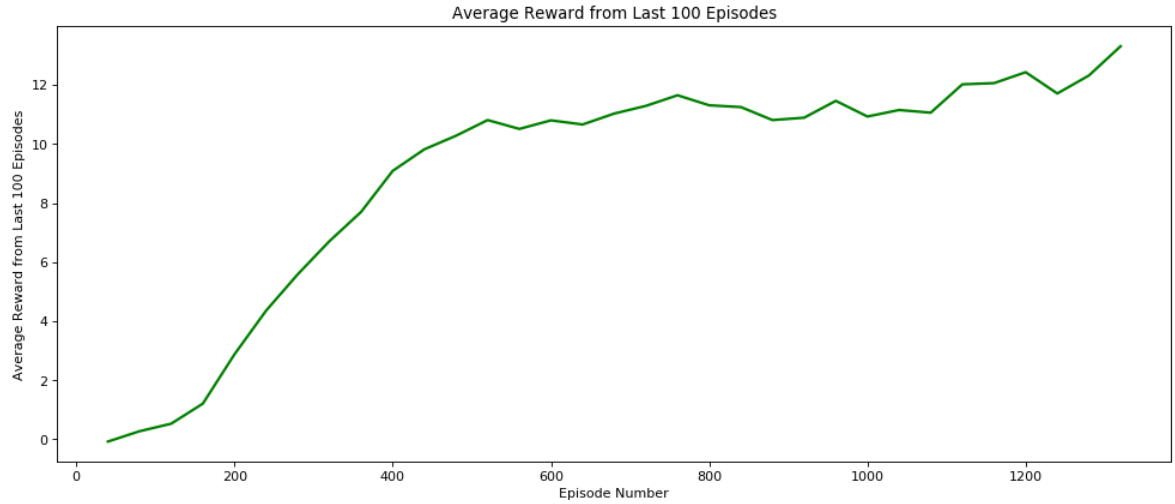
As mentioned in the introduction, the code is located in */notebook/colab/*. Again, **THE CODE DOESN'T WORK IN THE REPOSITORY CODEBASE.** That is why several **.html* version run in the Udacity Colab environment are added together with the saved modes' parameters.

The parameters used are the following:

```
EPS_DECAY = 0.995
GAMMA = 0.99

UPDATE_EVERY_STEP = 4
HARD_UPDATE_EVERY_STEPS = 4

ALPHA = 0.2
BETA_START = 0.6

ACTIONS_DIM = 1
MEMORY_SIZE = 2**14 # 2**14 = 16384
BATCH_SIZE = 64

EPS_START = 1.0
EPS_END = 0.01
```

The results from one of the three executions are the following:

```
Date and Time of Starting Execution: 04/02/2023 10:45:25
## Episode Number: 80, Average Score: 0.275, Duration[s], [mins]: 127.52, 2.13
## Episode Number: 160, Average Score: 1.21, Duration[s], [mins]: 121.83, 2.03
## Episode Number: 240, Average Score: 4.36, Duration[s], [mins]: 124.39, 2.07
## Episode Number: 320, Average Score: 6.71, Duration[s], [mins]: 126.96, 2.12
## Episode Number: 400, Average Score: 9.09, Duration[s], [mins]: 127.23, 2.12
## Episode Number: 480, Average Score: 10.28, Duration[s], [mins]: 128.52, 2.14
## Episode Number: 560, Average Score: 10.51, Duration[s], [mins]: 129.89, 2.16
## Episode Number: 640, Average Score: 10.66, Duration[s], [mins]: 129.45, 2.16
## Episode Number: 720, Average Score: 11.29, Duration[s], [mins]: 129.79, 2.16
## Episode Number: 800, Average Score: 11.31, Duration[s], [mins]: 130.06, 2.17
## Episode Number: 880, Average Score: 10.81, Duration[s], [mins]: 129.59, 2.16
## Episode Number: 960, Average Score: 11.46, Duration[s], [mins]: 128.93, 2.15
## Episode Number: 1040, Average Score: 11.15, Duration[s], [mins]: 128.84, 2.15
## Episode Number: 1120, Average Score: 12.02, Duration[s], [mins]: 128.97, 2.15
## Episode Number: 1200, Average Score: 12.43, Duration[s], [mins]: 129.06, 2.15
## Episode Number: 1280, Average Score: 12.32, Duration[s], [mins]: 129.12, 2.15
## Episode Number: 1325, Average Score: 13.59

## ENVIRONMENT SOLVED ##
  - Number of Episodes: 1225
  - Average Score: 13.59
  - Model Was Saved Into File: 2023-02-04-11-20-48_model_checkpoint.pth
```



Ideas for Future Work

Ideas for future work:

- Reimplement neural networks into TensorFlow, the framework I use more during my work.
- Finish full RAINBOW implementation.
- Study more complex neural network architectures together with the issue of overfitting.
- Create a better hyperparameter search schema.