Components of Agent

My personal goal for this task was to check improvements of the agent from a simple Q network closer to a full RAINBOW agent.

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 approach simulates a realistic set-up because these days it is very important to be fast and effective in getting to the functional solution.

AI Gym 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/00_environments.py* or in its frozen version in */docs/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/01_replay_buffer.py* or in its frozen version in */docs/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 simple and dueling networks were used.

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 target 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/ 02_double_q_agent_training.py*
- */notebook/final/02_double_q_agent_with_PER_training.py*

Sample runs are available in:

- */docs/ 02_double_q_agent_training.html.*
- */docs/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/testing*.

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.