Reinforcement Learning @FAU
Summer Term 2023

Sebastian Rietsch, Nico Meyer, Christopher Mutschler
2023-07-14 – Due 2023-07-19

Exercise 12
Discrete Batch-Constrained deep Q-Learning (BCQ)

This exercise will implement the discrete BCQ algorithm described in the paper ”Benchmarking Batch Deep Reinforcement Learning Algorithms” and employ it on the CartPole-v1 environment. For this, we first pre-train a standard DDQN agent (not neccessarily all the way to perfect behaviour) to generate offline data. The BCQ algorithm then is trained on this data, without the possibility of direct interaction with the environment – i.e. offline reinforcement learning.

Based on the skeleton, you should be able to complete this exercise with around three lines of code.

Programming Tasks: Inside `bcq.py`, you have to fill several core steps of the discrete BCQ algorithm. The routine for pre-training the agent and offline data generation is already implemented (but feel free to experiment with the hyperparameter settings).

1. Sample a batch of experience (i.e. observation, action, reward, consecutive state, termination flag) from the `offline_buffer`. The buffer is given as a `ReplayBuffer` object, see the `ddqn.py` file.
2. Compute the target Q-values (using the target network!), which is required in the loss function of the Q-value approximator:
$$\mathcal{L}(\theta) = \mathbb{E}\left[r + \gamma Q_{\theta'}(s', a') - Q_{\theta}(s, a)\right]$$
Note: Be careful when determining the action a', see also the next task.
3. Use the q-value and the imitation network to `predict` the action for a given state in a constrained neighborhood via
$$a' = \underset{G_{\omega}(a'|s')/max_{\tilde{a}} G_{\omega}(\tilde{a}|s') > T}{\operatorname{argmax}} Q_{\theta}(s', a'),$$
where T is the BCQ threshold and G_{ω} is a approximation of the behavioral policy (see the `BCQNetwork` class). Feel free to tune this thresholds, as it is a crucial for a good performance.

The performance of the BCQ agent is determined to a big part by the *quality* of the data it is fed. You can experiment with this pre-training the data-generating DDQN agent to different performances, for some examples see Figure 1.

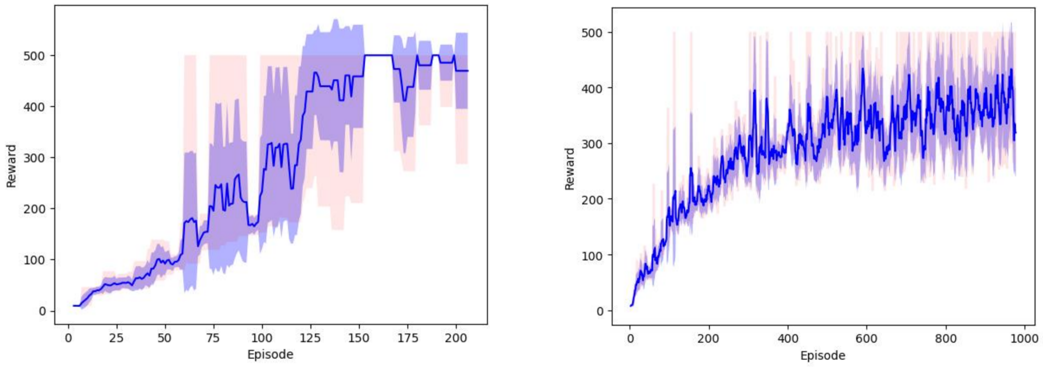

Figure 1: Performance of BCQ trained on data of different *quality*; (left) training with data generated following near-optimal policy (right) training with data from policy with average reward of 256.3