NetworkGym: Reinforcement Learning Environments for Multi-Access Traffic Management in Network Simulation (Supplementary Material)

Momin Haider
UC, Santa Barbara
momin@ucsb.edu

Ming Yin
Princeton University
my0049@princeton.edu

Menglei Zhang
Intel Labs
menglei.zhang@intel.com

Arpit Gupta
UC, Santa Barbara
arpitgupta@cs.ucsb.edu

Jing Zhu
Intel Labs
jing.z.zhu@intel.com

Yu-Xiang Wang
UC, San Diego
yuxiangw@ucsd.edu

We open-source our primary code and offline datasets at `github.com/hmomin/networkgym`. Each section (except Section 3) in this document references assets relative to the root directory of this repository.

1 Computational Resources

We make use of four internal 12 GB NVIDIA TITAN Xp GPUs to perform our experiments. With these GPUS, to perform all experiments described in this document requires roughly 1 month of compute, assuming each of 8 different CPU processes is used to perform an agent evaluation. Using only a single process to perform agent evaluation would result in the compute increasing to roughly 3 months.

2 Offline Data Collection

For each of three different heuristic policies (`throughput_argmax`, `system_default`, and `utility_logistic`), we collect and store 64 episodes of offline data on our NetworkGym Multi-Access Traffic Splitting environment (denoted `nqos_split`). Each episode contains 10,000 steps worth of data. The associated configuration file (located at `network_gym_client/envs/nqos_split/config.json`) for the episodes is chosen with the following constraints in mind:

- At initialization of each environment, four UEs are randomly stationed 1.5 meters above the x-axis between $x = 0$ and $x = 80$ meters. From there, they begin to bounce back and forth in the x-direction at 1 m/s for the entire duration of an episode.
- The Wi-Fi access points are stationed at $(x, z) = (30\text{m}, 3\text{m})$ and $(x, z) = (50\text{m}, 3\text{m})$, respectively.
- The LTE base station lies at $(x, z) = (40\text{m}, 3\text{m})$.
- The only change in the configuration file between episodes is the `random_seed` parameter. We use random seed values from 0 to 63, inclusive, for this parameter.

We store the resulting three offline datasets in the `NetworkAgent/buffers` directory. Each dataset is a folder that contains 64 `.pickle` files, one for each episode. Each `.pickle` file contains a tuple

Submitted to the 38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks. Do not distribute.

of four numpy arrays in the following order: (states, actions, rewards, next states) with shapes ([9999, 56], [9999, 4], [9999, 1], [9999, 56]), respectively.

We also provide a shell script (`offline_collection.sh`) to generate data for offline learning. The heuristic policy that takes actions in the environments can be specified at the top of the script.

3 Training Existing State-of-the-Art Offline RL Algorithms

To test several existing state-of-the-art offline reinforcement learning (RL) algorithms, we make use of the Clean Offline RL library provided at `github.com/tinkoff-ai/CORL`, which uses the Apache 2.0 license. More specifically, we modify their library at `github.com/hmomin/CORL-compare` to be compatible with our offline dataset generated on the NetworkGym simulator. The modifications we make to the offline RL algorithm files (located at `algorithms/offline`) only support the following purposes:

- We switch the algorithmic implementations from using D4RL-specific loading to using our NetworkGym `OfflineEnv` class instead.
- We remove all resulting unused D4RL-specific environment/dataset loading and evaluation code.
- We modify the `env` parameter in the `TrainConfig` class for each algorithm to use an environment specified by one of our three offline datasets.
- We modify the `normalize` boolean parameter (where applicable) in the `TrainConfig` class to toggle whether or not we would like the algorithm to perform feature normalization based on the offline dataset.

Using these modifications, any of the algorithm scripts at `algorithms/offline` can be executed directly to train these algorithms. We use the default hyperparameters for all algorithms, except where we toggle the `normalize` parameter.

4 Training PTD3

To train our implementation of Pessimistic TD3 (PTD3), we use the default hyperparameters in TD3+BC, except for the following modifications:

- We train PTD3 for 10,000 steps, instead of 1,000,000 steps, which we do for TD3+BC.
- We test PTD3 across various values of α and β; we then report the corresponding experimental results.

We provide the shell script `train_offline_ptd3.sh` to train PTD3 on any offline dataset generated by one of our heuristic algorithms. The desired values of offline dataset, α, and β can be specified at the top of the script.

5 Training Online Deep RL Algorithms

We use `stable-baselines3` to train two different online deep RL algorithms, PPO and SAC. We do so by initializing a random agent, then updating that agent through 8 successive phases. In each phase, we parallelize environment instantiations across 8 different random seeds, where each environment runs for 10,000 steps, resulting in a total of 64 different environment instantiations. In this way, the online learning algorithm trains across the same number of steps available in each of the offline datasets, to allow for proper comparison. Additionally, for our parallel environment random seeds, we use 0-7, inclusive, followed by 8-15, 16-23, ..., 56-63. We provide the shell script, `train_online_parallel.sh`, in order to perform this training process with PPO and SAC. We use the default hyperparameters specified by `stable-baselines3`.

6 Evaluating Trained Agents

Finally, to evaluate a trained agent (whether online or offline), we place the resulting model file in the `NetworkAgent/models` directory. Then, the model filename (without extension) can be specified as the `agent` parameter at the top of the `test_agent.sh` shell script and the script can be executed to evaluate the agent on a single 3,200 step episode. In our experiments, we evaluate each agent across 32 or 40 episodes (each with a different `random_seed` parameter), depending on the experiment. Each episode is 3,200 steps and the `random_seed` parameter takes on values between 128-159, inclusive, for 32 evaluation episodes or 128-167, inclusive, for 40 evaluation episodes. We otherwise use the same environment configuration details mentioned in Section 2.