Statistically Efficient Advantage Learning for Offline Reinforcement Learning in Infinite Horizons

Author Contributions Checklist Form

Part 1: Data

Abstract

The case study considered in this paper is implemented using the OhioT1DM dataset for blood glucose level prediction. The data contains continuous measurements for six patients with type 1 diabetes over eight weeks. The objective is to learn an optimal policy that maps patients' time-varying covariates into the amount of insulin injected at each time to maximize patients' health status.

Availability

The OhioT1DM dataset is available from http://smarthealth.cs.ohio.edu/OhioT1DM-dataset.html. However, a Data Use Agreement (DUA) is required to protect the data and ensure that it is used only for research purposes.

Description

In our experiment, we divide each day of follow-up into one hour intervals and a treatment decision is made every hour. We consider three important time-varying state variables, including the average blood glucose levels G_t during the one hour interval $(t-1, t]$, the carbohydrate estimate for the meal C_t during $(t-1, t]$ and Ex_t which measures exercise intensity during $(t-1, t]$. At time t, we define the action A_t by discretizing the amount of insulin In_t injected. The reward R_t is chosen according to the Index of Glycemic Control that is a deterministic function $G\_t + 1$. Detailed definitions of A_t and R_t are given as follows,

$$A_t = \begin{cases} 0, & \mathrm{In}_t = 0; \\ m, & 4m - 4 < \mathrm{In}_t \leq 4m \quad (m = 1, 2, 3); \\ 4, & \mathrm{In}_t > 12. \end{cases}$$

$$R_t = \begin{cases} -\frac{1}{30}(80 - G_{t+1})^2, & G_{t+1} < 80; \\ 0, & 80 \leq G_{t+1} \leq 140; \\ -\frac{1}{30}(G_{t+1} - 140)^{1.35}, & 140 \leq G_{t+1}. \end{cases}$$

Let $X\_t = (G\_t, C\_t, \mathrm{Ex}\_t)$. We define the state S_t by concatenating measurements over the last four decision points, i.e., $S\_t = (X\_t - 3, A\_t - 3, \cdots, X\_t)^\top$. This ensures the Markov assumption is satisfied. The number of decision points for each patient in the OhioT1DM dataset ranges from 1119 to 1288. Transitions across different days are treated as independent trajectories.

The data used in the paper is called the `trajs_mh.pkl`. However, for confidentiality considerations, we do not put it in this repository. The code to generate this data is placed in the `generate_trajs_mh.py` in the `data` folder. Once you have downloaded the raw data, put it in the same folder as the code, and then run the code to get the data used in this paper.

Synthetic data

Two OpenAI Gym environments,`LunarLander-v2` and `Qbert-ram-v0` are used to generate the synthetic data. You can run the `qr_dqn_online.ipynb` in the `data` folder and change the `env_name` and `num_actions` to get the `trajs_qr_dqn_lunar.pkl` and the `trajs_qr_dqn_qbert.pkl` respectively. These two files are zipped in the `data/synthetic_data.rar`.

Part 2: Code

Abstract

The `seal` folder contains the core code to implement the proposed method and various utility functions.

Description

seal package Overview

- `models`: network structures.
- `agents`: DQN, QR-DQN, MultiHeadDQN (REM), DiscreteBCQ, and BEAR(MMD replaced by KL_control) agents.
- `replay_buffers`: basic and prioritized replay buffers.
- `algos`: behavior cloning, density estimator, advantage learner, fitted Q evaluation, etc.
- `utils`: utility functions.

Computational Complexity

we assume that the forward and backward of network complexity is `S`.

- step 2 (Policy optimization): training `L` DQN agents, batch size `B_1`, training steps `I_1`, total `O(L * I_1 * B_1 * S)`
- step 3 (Estimation of the density ratio): training `L` density estimators, batch size `B_2`, training steps `I_2`, total `O(L * I_2 * B_2^4 * S)`
- step 4 (Construction of Pseudo Outcomes): batch size `B_3`, total `O(B_3 * N * T * A * S)`, where `N` number of trajs, `T` average length of trajs, `A` number of actions.
- step 5 (Training τ): batch size `B_4`, training steps `I_4`, total `O(I_4 * B_4 * S)`

Optional Information

All experiments run on a single computer instance with 40 Intel(R) Xeon(R) 2.20GHz CPUs.

- Python version: Python 3.6.8 :: Anaconda custom (64-bit)

Main packages for the proposed estimator

- numpy == 1.18.1
- pandas == 1.0.3
- sklearn == 0.22.1
- tensorflow == 2.1.0
- tensorflow-probability == 0.9
- gym == 0.17.3

Part 3: Reproducibility workflow

Synthetic data results

- copy the `data/data/synthetic_data.rar/trajs_qr_dqn_lunar.pkl` to the `lunarlander-v2/dqn_2_200/random/` folder, and change its name to `trajs_qr_dqn.pkl`. Use the `cd` command to switch to the `lunarlander-v2` directory and run the python file `batch_seal_vs_dqn.py` (around 20 hours without GPU support). This will generate DQN v.s. SEAL offline training results under 200 trajectories randomly sampled out of the total trajectories in `.csv` files. Similarly, we can obtain DDQN, QR-DQN, REM, Discrete-BCQ and Discrete-BEAR results. Same procedures to take with `Qbert-ram-v0`.
- We aggregate all the results in the `synthetic_results` folder with `plots_lunar` and `plots_qbert` folders. Each folder contains `dqn.csv`, `ddqn.csv`, `qrdqn.csv` and `4_methods.csv`.

Real data results

- Run the `DQN_mh.ipynb` under the `realdata` folder after putting the `trajs_mh.pkl` into the `realdata/data/mh/dqn` folder. This will generate DQN v.s. SEAL training results in a `.pkl` file. Similarly, we can obtain DDQN, QR-DQN, REM, Discrete-BCQ and Discrete-BEAR results.
- We aggregate all the results in the `real_data_results` folder containing `dqn.csv`, `ddqn.csv`, `qrdqn.csv` and `4_methods.csv`.

Figures

You can get the Figure 2, Figure 3 and Figure 4 in the article by running the file `plot_figures.py`. The `figs` folder contains these figures named `Figure_2.png`, `Figure_3.png` and `Figure_4.png`.

References

Thanks to Repos:

- https://github.com/google-research/batch_rl
- https://github.com/ray-project/ray
- https://github.com/sfujim/BCQ
- https://github.com/aviralkumar2907/BEAR