Towards Real-World Offline Reinforcement Learning

[Team: Offline Learners]

Anurag Koul
Ph.D. Computer Science, 6th year
koula@oregonstate.edu

Jeremy Dao
Ph.D. Computer Science, 3rd year
daoje@oregonstate.edu

Aseem Saxena
M.S Robotics, 1st year
saxena@oregonstate.edu

Ramya Jayaraman
M.S Robotics, 2nd year
jayaramr@oregonstate.edu

Abstract

Reinforcement Learning requires the entire model of the world or interactive access to the world. However, the world model may not be always known or it may be expensive or unsafe to perform multiple interactions with the world. In such scenarios, we would like to make use of existing transaction data to learn a control policy. This is addressed by a class of algorithms referred to as "Offline Reinforcement Learning". In this work, we study and implement "Behaviour Cloning"(BC), "TD3" and a combination of both "TD3+BC" for offline reinforcement learning. We evaluate them on various syntheic datasets and investigate the performance of each of them on different qualities of datasets. We also attempt to use offline RL for the real-world bipedal robot "Cassie" and introduce various datasets for a bipedal locomotion task.

1 Introduction

Reinforcement learning has shown very promising results in sequential decision making problems such as Chess, Go [22] and many other benchmarks [2]. This progress has been primarily accelerated due to the advancement in reinforcement learning algorithms and the marriage of deep learning with classic reinforcement learning [16]. An increase in the ease of access to simulators [2, 26], where the learning agent does online training via interaction with the virtual world, has also contributed to rapid advancement. However, agents trained in the virtual world are often times not transferable to the real-world due to approximation errors between the simulator and real world. This has led to the rise of algorithms that learn from a fixed dataset, collected by some arbitrary and possibly unknown process in the real-world. These algorithms are referred to as Offline Reinforcement Learning (also known as Batch Reinforcement Learning). This helps us learn policies from real-world transactions which could be collected via safe automated policies or a human operator.

Offline RL [12] comes with its own set of challenges. At the heart of the issue is the limited amount of data leading to an inability to do policy-evaluation for state-action pairs not covered within the dataset. Thereby, algorithms can only learn an optimal policy with coverage within the dataset. For unknown state-actions, we face extrapolation error that leads to erroneous value estimations for these states. The learned policies often over-estimates the value of these unknown states which disrupts policy improvement.

Objective. One of the primary solutions to the above problem involves restraining the policy learning to the actions covered in the dataset. There are various methods that have been proposed for the same issue: 1) KL control [8, 30] 2) Conservative Q Learning [11] 3) Behaviour constraint. In this work, we study one such algorithm “TD3+BC" [4], which is a model-free algorithm for learning a

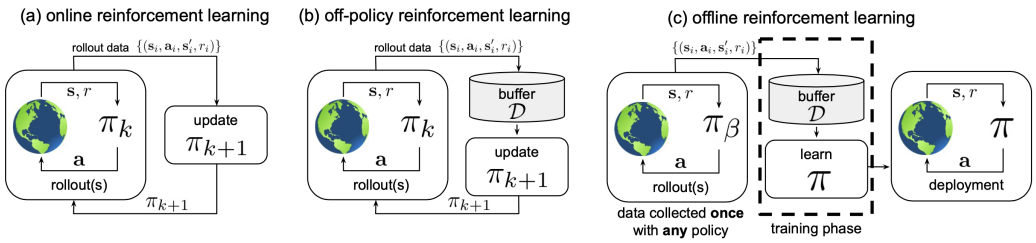

Figure 1: Schematic distinction between interactive RL and offline RL

control policy. This algorithm introduces a minimal change in the TD3 algorithm by introducing a behavior cloning (BC) constraint on policy updates. In this work, we progressively implement "TD3" [5], "BC", and then combine both of them to make "TD3+BC". Thereafter, we evaluate these algorithms on a set of environments in the D4RL dataset [3]. We also supplement our evaluation of this algorithm with a real-world application Cassie dataset.

2 Background

RL. We prelude discussion of our approach with a brief overview of Reinforcement Learning (RL) [27]. RL involves performing a sequence of actions in an environment and maximizing the cumulative return. The environment is defined by a Markov Decision Process (MDP) $< S, A, R, \delta, \gamma >$, indicating a finite state-space, finite action-space, bounded reward space, state-action transition matrix, and discount factor respectively. Given a policy π, which is a mapping from state to action selection distribution, we define the discounted return of a policy in state "s" after taking action "a" as

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{t=\infty} \gamma^t R_t \mid S_0 = s, A_t = a \right]$$

Offline RL. In figure[1] 1, we describe the progression from online to offline reinforcement learning. The primary assumption of on-policy RL [20] is the ability to interact with the environment to collect data for policy improvement. In this case, the exploration policy used is just the policy being learned, which is referred to as the target policy. This ensures that the coverage of the state-action space required by the policy is available. However, this limits our exploration to the target policy itself. In order to overcome this, we refer to off-policy RL, where the exploration policy and target policy are independent of each other. A good exploration policy encourages sufficient state-action space coverage for the off-policy updates.

In the case of offline RL, we no longer can interact with the environment and are limited to a fixed dataset (D). This is typically the case for real-world applications.

3 Algorithms

In this section, we introduce TD3 and Behaviour Cloning (BC), which when put together form the skeleton of the "TD3+BC" algorithm.

TD3. Twin Delayed DDPG (TD3) [5] is a model-free off-policy reinforcement learning algorithm which addresses the instabilties in training of the DDPG [14] algorithm. It can be used for both discrete as well as continuous action spaces. It introduces the following tricks:

1. **Clipped Double-Q Learning.** TD3 learns two Q-functions instead of one (hence "twin"), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.

[1]This image is borrowed from [12]

2. **"Delayed" Policy Updates.** TD3 updates the policy (and target networks) less frequently than the Q-function.
3. **Target Policy Smoothing.** TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q values along changes in the action.

BC. Behaviour cloning (BC) is simply imitation learning to replicate the data collection policy. This is particularly useful when an unknown expert policy is used for data collection and we need to replicate it. BC helps us stick to the action distribution in the dataset, thereby reducing extrapolation error during policy evaluation. It usually suffers when a collection of policies are used to collect a dataset, which is difficult to model using unimodal policy distribution classes.

TD3+BC. The fundamental offline-rl difficulty of out of distribution actions requiring extrapolation into the unknown state-action space can be addressed by regularizing the action selection via the KL divergence with the behaviour policy. Though this has been shown to be useful, TD3+BC argues that one can achieve a similar constaint in a much simplier manner by performing BC while learning a model-free policy as done in TD3. We share the entire algorithm in [1] and highlight the key idea over here. This algorithm involves only 2 minimal changes from the base TD3 algorithm:

1. Add a behaviour cloning regularization term while doing a TD3 policy update as shown in eq. (1)

$$\pi = \arg\max_{\pi} \mathbb{E}_{s\sim\mathcal{D}}\left[Q(s,\pi(s))\right] \rightarrow \pi = \arg\max_{\pi} \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\lambda Q(s,\pi(s)) - (\pi(s)-a)^2\right]. \tag{1}$$

2. Normalization of every state-feature (s_i) in the dataset as shown in eq. (2); where μ_i, σ_i, ϵ are the mean, standard deviation of the ith state-feature across the dataset, and small normalization constant respectively.

$$s_i = \frac{s_i - \mu_i}{\sigma_i + \epsilon}, \tag{2}$$

Algorithm 1 TD3+BC

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \arg\min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s,a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_{a^\star} \lambda Q_{\theta_1}(s, a^\star) - (a^\star - a)^2|_{a^\star=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
  end if
end for

4 Experiments

In this work, we want to ask ourselves *3 questions* by replicating the TD3+BC algorithm:

- Is it possible to learn policies using offline datasets?
- What is the effect of different qualities of the dataset on offline learning?
- Does learning with real-world data eliminate the issue of sim-to-real transfer?

4.1 Architecture & Implemenation Details.

In all our experiments, we use an actor-critic arcitecture where both actor and critic are seperate networks, each having 2 fully-connected layers with ReLU intermediate activation function. Each hidden-layer has 256 neurons. The last layer of the actor network has dimensions equivalent to the cardinality of the action-space of an environmnet. The critic network receives the current state and predicted action by the actor as an input. It has a unit neuron in the last layer predicting the q-value corresponding to the input. Also, in "BC" experimenets, we only use the actor network. We use the Adam optimizer [9] with learning rate of 0.0003 for the critic and 0.0001 for the actor. Also, we use a discount factor (γ) of 0.99 in all the cases and report results over multiple runs for the above hyper-parameters. We did not performed any hyper-parameter tuning in our case due to resource constraints.

4.2 Synthetic Tasks: D4RL

Task Description. D4RL is a collection of synthetic environments [3] based on OpenAI gym [2] and the MuJoCo physics simulator [28]. It provides a collection of varying qualities of datasets for each of the environments. More information of these datasets can be found at `https://sites.google.com/view/d4rl/`. For our work we will focus on just two of the included environments:

- **Maze2D:** A grid with 4 levels of complexity: 1) open 2) umaze 3) medium and 4) hard, listed in the increasing order of complexity. The objective of the environment is to navigate a 2d agent to a fixed goal location. The initial position of the 2d-agent is randomly placed in the grid. We only have one kind of dataset for this environment which we refer as *"random-goal navigation"*. This dataset is generated by randomly navigating to different goal locations in the maze. This leads to good coverage of the state and action space.
- **Gym-Mujoco:** This includes a collection of 3 classic control tasks in OpenAI gym: 1) Half-Cheetah 2) Hopper and 3) Walker2d. These environments comprise of a single task of maximizing the distance covered by the agent. 5 datasets of different qualities are provided for each these environments. This includes a *"random", "medium"* and *"expert"* dataset which comprises of trajectories generated by a random policy, medicore policy, and optimal policy, respectively. There is also a *"medium-replay"* and *"medium-expert"* dataset which comprises of medicore policy trajectories mixed with exploration data in a replay buffer and expert trajectories, respectively.

Maze2d Analysis. We present our results for Maze2d environment in figure [4], having normalized scores on the x-axis and update steps on the y-axis. We observe that "BC" tends to have the worst performance in all maze configurations. This is primarly due to the nature of the dataset which comprises of random navigation in the maze, producing random behaviour. Thus a deterministic policy that learns to capture such random behaviour will not actually reach the goal very often. We also obsereve "TD3" (also refered as TD3(offline)) tends to perform fairly well compared to "BC". This is primarly due to good coverage of state-action space in the "random-goal navigation" dataset, leading to coveragance towards a "near-optimal" policy. In this case, the combination of "TD3+BC" tends to hurt the converagence as the "BC" constraint will deviate the policy towards randomness.

Gym-Mujoco Analysis. We share our results for the ym-mujoco environments in figure [5]. Each row belongs to a specific environment and each column belongs to a specific dataset quality. Begining with "expert" quality dataset, we observe that "BC" tends to converage the fastest to the optimal policy, as this is a dataset of optimal trajectories. "TD3+BC" also converages to the optimal behaviour

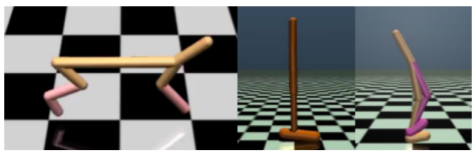

Figure 2: Gym-Mujoco Tasks

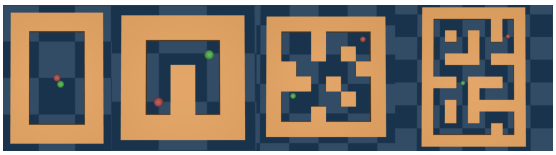

Figure 3: Maze2d Tasks

but tends to have a slow converagance. This is interesting and shows the importance of having regularization(λ) over Q-values losses rather than an imitation loss as in eq (1). Also, “TD3(offline)" tends to fail completely in this case due to limited state-action coverage. This primarly is due to over-estimation issues for out-of-distribution states and actions which lead the learned policy to unknown spaces where the behaviour is no longer defined. As we move from “medium" to “medium-replay" dataset, we observe “TD3+BC" performing better than “BC". This is likely because “BC" tries to imitate a worse medicore policy as we move from medium to medium-replay (which contains random exploration data) and multi-modal policies are difficult to capture with a single deterministic policy. Finally, we look at "random" dataset and find that all algorithms tend to perform poorly as compared to earlier datasets. This is opposite to our observation in the “Maze2d" case as for “Gym-Mujoco" a collection of random behaviour leads to coverage of only a small state and action space due to high dimensionality of both the observation and action space.

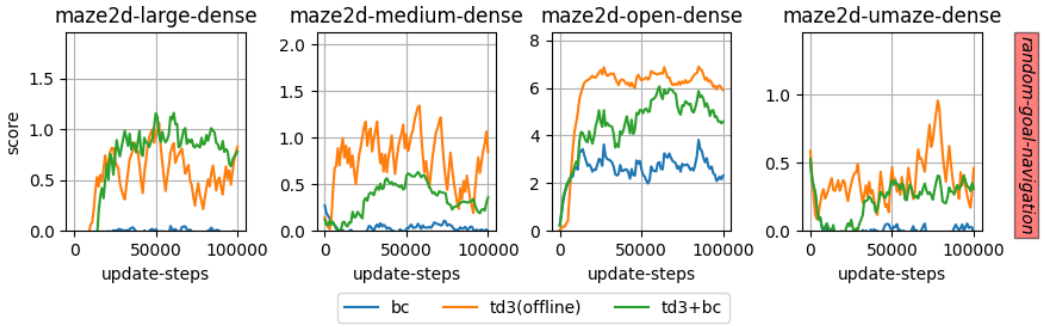

Figure 4: Performance comparison of various algorithms 1) BC 2) TD3(offline) and 3) TD3+BC in various maze-configurations of maze2d domain. There is only one kind of dataset in this case.

4.3 Real World: Cassie

Offline RL also presents some incredible potential benefits for robotic learning. Many research works utilizing RL for robotic control use very sample inefficient on-policy gradient methods [21, 29, 25, 13]. As such, learning usually happens in simulation. However, this introduces what is known as the “sim-to-real" gap, where errors in the simulation model cause learned policies to act differently on hardware versus in simulation. There have been much research devoted to crossing this gap, such as using dynamics randomization [18] or optimization of the estimated model parameters [1]. Others try to sidestep the problem entirely by learning on physical hardware [31, 7]. However, due to the sample inefficiency of RL methods, gathering the large amount of data required on hardware can be difficult or be extremely expensive [6]. Offline RL presents a possible solution to such limitations. Learning from a single static dataset would dramatically reduce the number of samples required while allowing for hardware learning to successfully realize learned policies on hardware. In order to study such an application, we introduce two kinds of datasets: a *simulation* dataset in order to evaluate the effectiveness of an offline RL method for a locomotion task, and a *hardware* dataset to evaluate if offline RL can learn a policy suitable for hardware execution. Both datasets will be generated with the same previously trained expert walking policy capable of successful, robust walking up to 3.0 m/s.

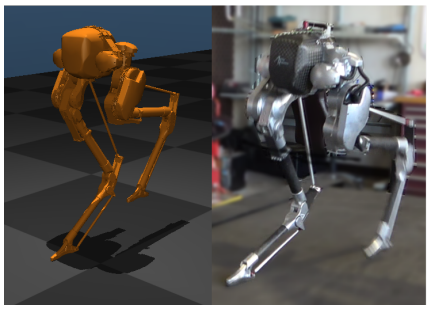

Figure 6: Simulator (left) and Real (right) Bipedal Cassie Robot

Simulation Datasets: To collect the simulation dataset we sweep a range of walking speeds from 0 to 3.0 m/s and turning speeds from 0 to $\frac{\pi}{8}$ rad/s and collect 5 gait cycles for each command. To further diversify the data, we collect trajectories where the robot is perturbed slightly by a small force in varying directions. Note that the size of the perturbation force is small enough such the policy

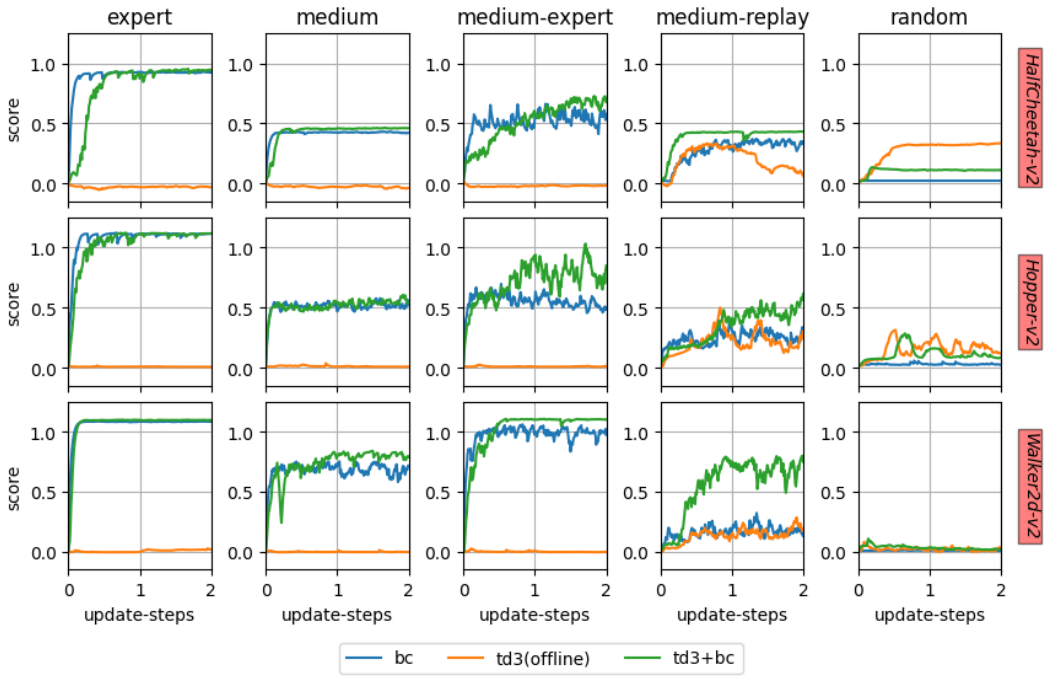

Figure 5: Performance comparison of various algorithms 1) BC 2) TD3(online) and 3) TD3+BC in various gym-mujoco environments. Here, the x-axis shows the normalized score and the y-axis shows the number of 100,000 updates-steps. For each environment, we evaluate over 5 kinds of datasets corresponding to each column. Also, results are aggregated over 3 runs.

can resist it without falling down, but large enough to require a change in the gait to stabilize. This ensures that we do not collect repeats of the same trajectory due to the cyclic nature of walking. This simulation based dataset serves as our “easy" test case for learning a Cassie policy with TD3+BC. Using simulation data also allow us to directly use true simulation and reward data.

Hardware Dataset: Opposed to this is the hardware version of the dataset. This data uses the same policy and the same set of speed and turning commands to collect similar trajectories. However, since this data is collected on hardware, the data is *not* ground truth and is the output of a state estimator. This noise and error in the state data makes it more difficult for a policy to learn as it does not know the true dynamics of how some action will affect the real state of the robot. Furthermore, in order to perform offline RL, our offline dataset needs to have state, actions, and *rewards*. Unfortunately further adding to the difficulties, there are some reward function components that we cannot measure accurately on hardware. In simulation where the original expert policy is learned, we have access to all ground truth information about the robot state. However, this is not true on hardware. For example, as the foot lacks any force sensor, ground reaction force at the foot can only be roughly estimated on the physical robot through deflection of the knee spring and does not provide an accurate measurement. Global foot velocity is always required for our reward calculation, but requires some grounding in the global frame like estimation of where the ground is, which can be difficult in locomotion as contact is made and broken constantly with alternating feet. To successfully use this dataset for offline RL we will have to work around this challenge.

Offline RL has begun to show success when applied to robotic control tasks [10, 19, 15, 17], and several previous works attempt to solve this reward problem in a number of ways. For example, [32] use unlabeled data to learn a reward function from which to perform offline RL on. This allows them to use teleoperation demonstrations which have no pre-described reward value for learning. Due to learning a reward function, this method proved to be more effective than just imitation learning. Rather than just learning to copy the demonstration data, the policy will actually optimize some (hopefully) meaningful reward function. Other works use a simple sparse reward for task completion

to learn robotic skills [23]. Sinha et. al. [24] attempt to robustify offline RL by augmenting the static dataset to improve out-of-distribution generalization. Such a technique could allow us to use our inaccurate reward estimates on hardware while still learning a successfull policy. For ease and simplicity, in our work we just use inaccurate measurements from the state estimator as reward signals and hope that they are close enough to the true reward that the learned policy will reflected the desired true motion.

Analysis. We share our results for cassie experiements in fig. [8]. In the case of simulator datasets, Behaviour Cloning tends to recover functioning policies for “expert", “speed-only" and “sweep-half" datasets. BC in the “turn-only" dataset has sub-optimal performance as the dataset only includes standing and rotation around the same spot by the robot. Surprsingly, “TD3" and “TD3+BC" tend to fail in all the cases, despite regularization being applied to the Q-value estimation in eq. (1). We conjecture this due to a low amount of data and high dimensionality of the environment. Though we have sub-optimal policies using "BC", we attempted to transfer them to the hardware and found them to be very unstable and untransferable. Again, we believe that this is due to insufficient amount of data leading to a very sensitive policy. If the policy does not start in a state it has seen before, it will never stabilize and will quickly fail. This sensitivity is exacebated on hardware where sim-to-real differences mean that the policy may never see an exact same state that it was trained on.

This encouraged us to simply train over the actual, but limited, hardware data and validate if this would bridge the gap of the inaccurate simulator. Unfortunately, upon evaluation, we observed learning on this dataset to not converage to a stable policy, probably due to the small amount of data. We attempted to overcome this by inducing mixture of simulator data with the hardware data. Despite this, none of the policies improved as shown in fig. 8 row 2 and row 3.

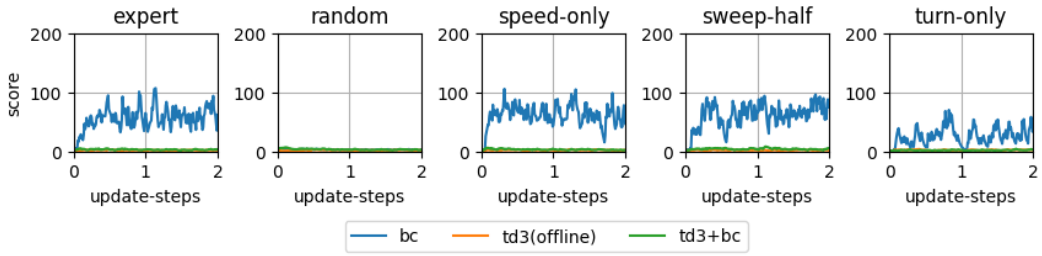

Figure 7: Performance Comparison of various algorithms 1) BC 2) TD3(offline) and 3) TD3+BC in *simulation* datasets for cassie. In these case, evaluation was done in simulator itself.

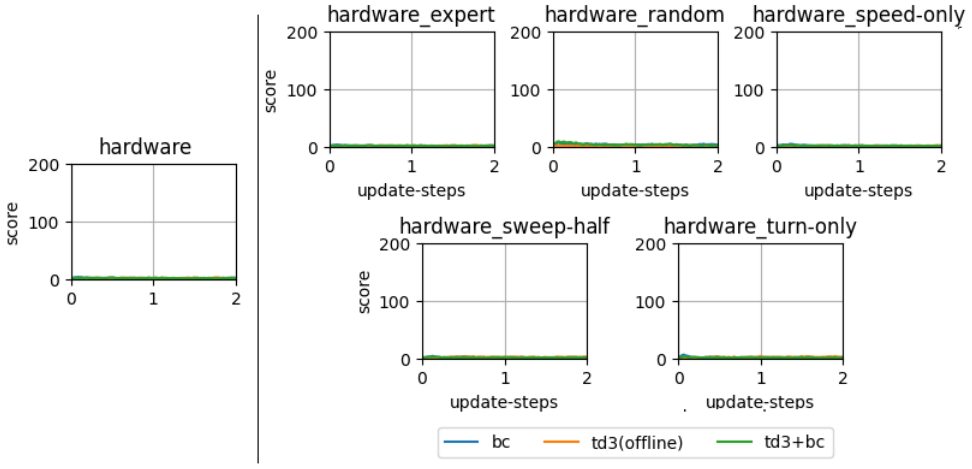

Figure 8: Performance Comparison of various algorithms 1) BC 2) TD3(offline) and 3) TD3+BC in *hardware* dataset (right) and "*mixture* of hardware data with *simulator* data" (right) for cassie.

5 Summary

We investigated “BC", “TD3", and “TD3+BC" for offline reinforcement learning over various environments and different qualities of datasets. Overall, we observed “BC" should be preferred whenever we have expert to medium quality datasets, “TD3" should be used whenever we have sufficiently large state-action space coverage, and finally, “TD3+BC" is an intermediatory to both of them and should be preferred when we have medium quality dataset with small coverage. To our surprise, limited data of the real-world is still a bottleneck for directly learning the policies with offline RL algorithms. In the future, we would like to investigate data-augmentation and multi-quality mixture of simulated datasets with the real-world dataset to see if it improves policy learning.

6 Task Division

In the following, we share the core responsibilities of each project member. At the same time, we emphasize that each member collaboarted on different tasks of this project.

- **Anurag**: Implementation , evaluation and analysis with synthetic *D4RL* environments for TD3 and TD3+BC.
- **Jeremy**: Collect, evaluate, and analyze *real-word* Cassie dataset with TD3+BC.
- **Aseem**: Collect, evaluate, and analyze *simulation* Cassie dataset with TD3+BC. He also investigated CQL[11] algorithm a series of other algorithms results of which couldn’t be shared due to computational constraints.
- **Ramya**: Implementation of Behavior Cloning and integration with TD3 to create TD3+BC.

Code for our project can be downloaded at `https://drive.google.com/file/d/1TCaQ70OPdi1NwMaSOmj7Pr1Vwf_kk6_D/view?usp=sharing`

References

[1] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning Agile Robotic Locomotion Skills by Imitating Animals. 2020.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[4] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021.

[5] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[6] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. IEEE, may 2017.

[7] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to Walk via Deep Reinforcement Learning. 2018.

[8] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Aviral Kumar, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine. A Workflow for Offline Model-Free Robotic Reinforcement Learning. (CoRL):1–28, 2021.

[11] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[12] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[13] Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots. mar 2021.

[14] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[15] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Dieter Fox. IRIS: Implicit Reinforcement without Interaction at Scale for Learning Control from Offline Robot Manipulation Data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4414–4420. IEEE, may 2020.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[17] Suraj Nair, Eric Mitchell, Kevin Chen, Brian Ichter, Silvio Savarese, and Chelsea Finn. Learning Language-Conditioned Robot Behavior from Offline Data and Crowd-Sourced Annotation. pages 1–23, 2021.

[18] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810. IEEE, may 2018.

[19] Rafael Rafailov, Tianhe Yu, Aravind Rajeswaran, and Chelsea Finn. Offline Reinforcement Learning from Images with Latent Space Models. 144:1–15, 2020.

[20] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

[21] Jonah Siekmann, Yesh Godse, Alan Fern, and Jonathan Hurst. Sim-to-Real Learning of All Common Bipedal Gaits via Periodic Reward Composition. nov 2020.

[22] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[23] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning. (CoRL):1–16, 2020.

[24] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. S4RL: Surprisingly Simple Self-Supervision for Offline Reinforcement Learning. pages 1–13, 2021.

[25] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. In *Robotics: Science and Systems XIV*, Pittsburgh, Pennsylvania, jun 2018. Robotics: Science and Systems Foundation.

[26] Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control, 2020.

[27] Sebastian Thrun and Michael L Littman. Reinforcement learning: an introduction. *AI Magazine*, 21(1):103–103, 2000.

[28] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[29] Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. DeepGait: Planning and Control of Quadrupedal Gaits Using Deep Reinforcement Learning. *IEEE Robotics and Automation Letters*, 5(2):3699–3706, apr 2020.

[30] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[31] Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani. Data Efficient Reinforcement Learning for Legged Robots. (CoRL):1–10, 2019.

[32] Konrad Zolna, Alexander Novikov, Ksenia Konyushkova, Caglar Gulcehre, Ziyu Wang, Yusuf Aytar, Misha Denil, Nando de Freitas, and Scott Reed. Offline Learning from Demonstrations and Unlabeled Experience. pages 1–13, 2020.