Journal of Machine Learning Research 23 (2022) 1-20 Submitted 1/22; Revised 9/22; Published 10/22

d3rlpy: An Offline Deep Reinforcement Learning Library

Takuma Seno SENO@AILAB.ICS.KEIO.AC.JP
Keio University
Kanagawa, Japan
Sony AI
Tokyo, Japan

Michita Imai MICHITA@AILAB.ICS.KEIO.AC.JP
Keio University
Kanagawa, Japan

Editor: Alexandre Gramfort

Abstract

In this paper, we introduce `d3rlpy`, an open-sourced offline deep reinforcement learning (RL) library for Python. `d3rlpy` supports a set of offline deep RL algorithms as well as off-policy online algorithms via a fully documented plug-and-play API. To address a reproducibility issue, we conduct a large-scale benchmark with D4RL and Atari 2600 dataset to ensure implementation quality and provide experimental scripts and full tables of results. The `d3rlpy` source code can be found on GitHub: `https://github.com/takuseno/d3rlpy`.

Keywords: offline reinforcement learning, deep reinforcement learning, reproducibility, open source software, pytorch

1. Introduction

Deep reinforcement learning (RL) has been led to significant advancements in numerous domains such as gaming (Wurman et al., 2022) and robotics (Lee et al., 2020). While RL algorithms have a potential to solve complex tasks, active data collection is a major challenge especially for environments where interaction is expensive. Offline RL (Levine et al., 2020), where algorithms find a good policy within a previously collected static dataset, has been considered as a solution to this problem.

Although recent offline deep RL papers are published with author-provided implementations, they are scattered across different repositories and do not provide standardized interfaces, which makes it difficult for researchers to incorporate the algorithms into their projects. It is also crucial for researchers to have an access to faithfully benchmarked implementations to deal with a reproducibility problem (Henderson et al., 2017).

In this paper, we introduce `Data-Driven Deep Reinforcement Learning library for Python` (`d3rlpy`), an offline deep RL library for Python. `d3rlpy` provides a set of off-policy offline and online RL algorithms built with `PyTorch` (Paszke et al., 2019). API of all implemented algorithms is fully documented, plug-and-play and standardized so that users can easily start experiments with `d3rlpy`. To solve the reproducibility issue in offline RL, a large-scale faithful benchmark is conducted with `d3rlpy`.

©2022 Takuma Seno and Michita Imai.

License: CC-BY 4.0, see `https://creativecommons.org/licenses/by/4.0/`. Attribution requirements are provided at `http://jmlr.org/papers/v23/22-0017.html`.

2. Related work

The choice of API design determines user experience, which has to balance the tradeoff between ease of use and flexibility. `KerasRL` (Plappert, 2016) and `Stable-Baselines3` (Raffin et al., 2021) provide deep RL algorithms with plug-and-play API and extensive documentations. `Tensorforce` (Kuhnle et al., 2017), `MushroomRL` (D'Eramo et al., 2021) and `Tianshou` (Weng et al., 2021) provide moduralized deep RL components that allow users to conduct custom experiments. `SaLinA` (Denoyer et al., 2021) provides a general framework for decision-making agents where users can implement scalable algorithms on top of it. To encourage a broader RL community to start offline RL research, `d3rlpy` is designed to be the first library that privdes fully documented plug-and-play API for offline RL experiments.

From the reproducibility perspective, `ChainerRL` (Fujita et al., 2021) and `Tonic` (Pardo, 2020) provide many deep RL algorithms with faithful reproduction results. `Dopamine` (Castro et al., 2018) is also the widely used implementation in the community, which focuses on making DQN-variants (Hessel et al., 2018) available for researchers. `d3rlpy` is also the first library accompanied by a number of offline RL algorithms and the extensive benchmark results in this research field.

Integrating the fully documented plug-and-play API and a number of faithfully benchmarked offline RL algorithms altogether, it is difficult for the existing libraries to instantly achieve the same offline RL research experience as `d3rlpy`.

3. Design of d3rlpy

In this section, the library design of `d3rlpy` is described.

3.1 Library interface

```python
import d3rlpy

# prepare MDPDataset object
dataset, env = d3rlpy.datasets.get_dataset("hopper-medium-v0")
# prpare Algorithm object
sac = d3rlpy.algos.SAC(actor_learning_rate=3e-4, use_gpu=0)
# start offline training
sac.fit(
    dataset,
    n_steps=500000,
    eval_episodes=dataset.episodes,
    scorers={"environment": d3rlpy.metrics.evaluate_on_environment(env)},
)
# seamlessly start online training
sac.fit_online(env, n_steps=500000)
```

`d3rlpy` provides scikit-learn-styled API (Pedregosa et al., 2011) to make the use of this library as easy as possible. In terms of the library design, there are two main differences from the existing libraries. First, `d3rlpy` has an interface for offline RL training, which takes a dedicated RL dataset component, `MDPDataset` described in Section 3.2. Second, all methods for training such as `fit` and `fit_online` are implemented in `Algorithm` components to make `d3rlpy` as plug-and-play as possible. For the plug-and-play user experience, neural network architectures are automatically selected from MLP and the convolutional model (Mnih et al.,

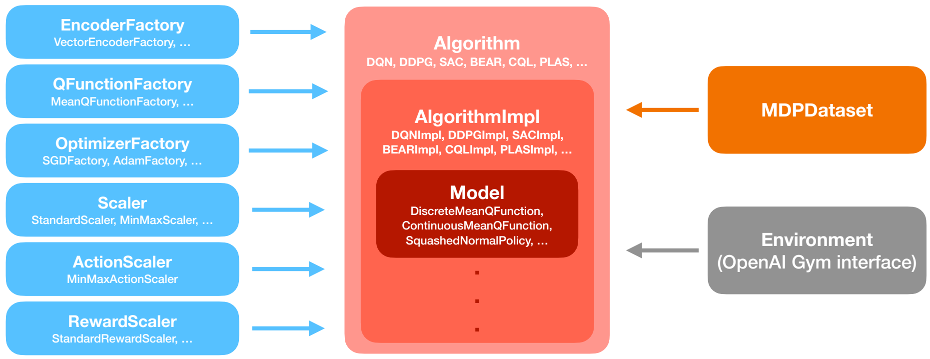

Figure 1: The illustration of module components in `d3rlpy`. `MDPDataset` and OpenAI Gym-styled environment can be used to train policies.

2015) depending on observation, which allows users to start training without composing neural network models unless using customized architectures described in Section 3.2. These design choices are expected to lower the bar to start using this library.

Since `d3rlpy` supports both offline and online training, the seamless transition from offline training to online fine-tuning is realized. Fine-tuning policies trained offline is demanded, but is still a challenging problem (Nair et al., 2020; Kostrikov et al., 2021). This seamless transition supports the further research by allowing RL researchers to easily conduct fine-tuning experiments.

3.2 Components

We highlight main components provided by `d3rlpy`. Figure 1 depicts module components in `d3rlpy`. All of these components provide the standardized API. The full documentation including extensive tutorials of the library is available at `https://d3rlpy.readthedocs.io`.

Algorithm. `Algorithm` components provide the offline and online training methods described in Section 3.1. `Algorithm` is implemented in a hierarchical design that internally instantiates `AlgorithmImpl`. This hierarchy is to provide high-level user-friendly API such as `fit` method for `Algorithm` and low-level API such as `update_actor` and `update_critic` for `AlgorithmImpl`. The main motivation of this hierarchical API system is to increase module resusability of the algoirthms when the new algorithm only requires high-level changes. For example, *delayed policy update* of TD3, which updates policy parameters every two gradient steps, can be implemented by adjusting frequency of `update_actor` method calls in the high-level module without changing the low-level logics.

MDPDataset. `MDPDataset` provides the standardized offline RL dataset interface. Users can build their own dataset by using logged data consisting with numpy arrays (Harris et al., 2020) of `observations`, `actions`, `rewards`, `terminals` and optionally `timeouts` (Pardo et al., 2018). The popular benchmark datasets such as D4RL and Atari 2600 datasets are also provided by `d3rlpy.datasets` package that converts them into `MDPDataset` object. In addition, `d3rlpy` supports automatic data collection by giving OpenAI Gym (Brockman et al., 2016) style environment, which exports collected data as `MDPDataset` object. For diverse sets of dataeset creation, the data collection can be performed with and without parameter updates.

EncoderFactory. User-defined custom neural network models are supported via `EncoderFactory` components. Users can define all function approximators to train in `d3rlpy` by building their own `EncoderFactory` components. This flexibility allows the use of the complex architectures (He et al., 2016) and experiments with partially pretrained models (Shah and Kumar, 2021).

QFunctionFactory. `d3rlpy` provides `QFunctionFactory` components that allow users to use distributional Q-functions: Quantile Regression (Dabney et al., 2018b) and Implicit Quantile Network (Dabney et al., 2018a). The distributional Q-functions dramatically improve performance by capturing variance of returns. Unlike conventional RL libraries that implement distributional Q-functions as DQN-variants, `d3rlpy` enables users to use them with all implemented algorithms, which reduces complexity to support algorithmic-variants such as QR-DQN (Dabney et al., 2018b) and a discrete version of CQL (Kumar et al., 2020).

Scaler, ActionScaler and RewardScaler. By exploiting static dataset in offline RL training, `d3rlpy` provides various preprocessing and postprocessing methods through `Scaler`, `ActionScaler` and `RewardScaler` components. For observation preprocessing, *normalization*, *standardization* and *pixel* are available. The observation `standardization` has been shown to improve policy performance in offline RL setting (Fujimoto and Gu, 2021). Regarding action preprocessing, *normalization* is available and action output from a trained policy is denormalized to original scale as postprocessing. Lastly, reward preprocessing supports *normalization*, *standardization*, *clip* and *constant multiplication*.

4. Large-scale benchmark

To address the reproducibility problem (Henderson et al., 2017), the implemented algorithms[1] are faithfully benchmarked with D4RL (Fu et al., 2020) and Atari 2600 datasets (Agarwal et al., 2020). The full Python scripts used in this benchmark are also included in our source code [2], which allows users to conduct additional benchmark experiments. Full tables of benchmark results are reported in Appendix A, Appendix B and Appendix C. The all logged metrics are released in our GitHub repository [3].

5. Conclusion

In this paper, we introduced an offline deep reinforcement learning library, `d3rlpy`. `d3rlpy` provides a set of offline and online RL algorithms through the standardized plug-and-play API. The large-scale faithful benchmark was conducted to address the reproducibility issue.

1. `d3rlpy` implements NFQ (Riedmiller, 2005), DQN (Mnih et al., 2015), Double DQN (van Hasselt et al., 2015), DDPG (Lillicrap et al., 2016), TD3 (Fujimoto et al., 2018), SAC (Haarnoja et al., 2018), BCQ (Fujimoto et al., 2019b), BEAR (Kumar et al., 2019), CQL (Kumar et al., 2020), AWAC (Nair et al., 2020), CRR (Wang et al., 2020), PLAS (Zhou et al., 2020), PLAS+P (Zhou et al., 2020), TD3+BC (Fujimoto and Gu, 2021) and IQL (Kostrikov et al., 2021).
2. `https://github.com/takuseno/d3rlpy/tree/master/reproductions`
3. `https://github.com/takuseno/d3rlpy-benchmarks`

Acknowledgments

This work is supported by Information-technology Promotion Agency, Japan (IPA), Exploratory IT Human Resources Project (MITOU Program) in the fiscal year 2020. We would like to express our genuine gratitude for the contributions made by the voluntary contributors. We would also like to thank our users who have provided constructive feedback and insights.

Appendix A. Benchmark Results: D4RL

We evaluated the implemented algorithms on the open-sourced D4RL benchmark of OpenAI Gym MuJoCo tasks (Brockman et al., 2016; Fu et al., 2020). We followed the experimental procedure described in Fu et al. (2020). We trained each algorithm for 500K gradient steps and evaluated every 5000 steps to collect evaluation performance in environments for 10 episodes. Table 1 shows hyperparameters used in benchmarking. We used the same hyperparameters as the ones previously reported in previous papers or recommended in author-provided repositories. We used discount factor of 0.99, target update rate of 5e-3 and an Adam optimizer (Kingma and Ba, 2014) across all algorithms. The default architecture was MLP with hidden layers of [256, 256] unless we explicitly address it. We repeated all experiments with 10 random seeds.

Table 2 shows results of the benchmark in normalized scale (Fu et al., 2020). Table 3, 4, 5, 6, 7, 8, 9, 10 and 11 show side-by-side comparisons with reference scores. CRR is not included in this side-by-side comparison because CRR has not been benchmarked with D4RL and an author-provided implementation is not publically available. Considering the fact that standard deviations for some algorithms were not reported by the authors and many algorithms in offline RL research were originally benchmarked with only 3 or 4 random seeds, we believe that performance discrepancy is trivial.

Algorithm	Hyperparameter	Value
SAC (Haarnoja et al., 2018)	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	256
AWAC (Nair et al., 2020)	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	1024
	λ (Nair et al., 2020)	1
	Actor hidden units	[256, 256, 256, 256]
	Actor weight decay	1e-4
	Critic hidden units	[256, 256, 256, 256]
BCQ (Fujimoto et al., 2019b)	Critic learning rate	1e-3
	Actor learning rate	1e-3
	VAE learning rate	1e-3
	Mini-batch size	100
	λ (Fujimoto et al., 2019b)	0.75
	Critic hidden units	[400, 300]
	Actor hidden units	[400, 300]
	VAE encoder hidden units	[750, 750]
	VAE decoder hidden units	[750, 750]
	VAE latent size	$2 \times \lvert A \rvert$
	Perturbation range	0.05
	Action samples	100
BEAR (Kumar et al., 2019)[4]	Critic learning rate	3e-4
	Actor learning rate	1e-4
	VAE learning rate	3e-4

	α learning rate	1e-3
	Mini-batch size	256
	VAE encoder hidden units	[750, 750]
	VAE decoder hidden units	[750, 750]
	VAE latent size	$2 \times \|A\|$
	MMD σ	20
	MMD kernel	laplacian (gaussian for HalfCheetah)
	MMD action samples	4
	α threshold	0.05
	Target action samples	10
	Evaluation action samples	100
	Pretraining steps	40000
CQL (Kumar et al., 2020)[5]	Critic learning rate	3e-4
	Actor learning rate	1e-4
	Fixed α	5 (10 for medium datasets)
	Mini-batch size	256
	Critic hidden units	[256, 256, 256]
	Actor hidden units	[256, 256, 256]
	Action samples	10
CRR (Wang et al., 2020)	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	256
	Action samples	4
	Advantage type	mean
	Weight type	binary
IQL (Kostrikov et al., 2021)	Critic learning rate	3e-4
	Actor learning rate	3e-4
	V-function learning rate	3e-4
	Mini-batch size	256
	Expectile	0.7
	Inverse temperature	3.0
	Actor learning rate scheduler	Cosine
PLAS (Zhou et al., 2020)	Critic learning rate	1e-3
	Actor learning rate	1e-4
	VAE learning rate	1e-4
	Mini-batch size	100
	Critic hidden units	[400, 300]
	Actor hidden units	[400, 300]
	VAE encoder hidden units	[750, 750] ([128, 128] for medium-replay)
	VAE decoder hidden units	[750, 750] ([128, 128] for medium-replay)
	λ (Fujimoto et al., 2019b)	1.0
	VAE latent size	$2 \times \|A\|$
	VAE pretraining steps	500000
PLAS+P (Zhou et al., 2020)	Critic learning rate	1e-3
	Actor learning rate	1e-4
	VAE learning rate	1e-4
	Mini-batch size	100
	Critic hidden units	[400, 300]
	Actor hidden units	[400, 300]
	VAE encoder hidden units	[750, 750] ([128, 128] for medium-replay)
	VAE decoder hidden units	[750, 750] ([128, 128] for medium-replay)
	λ (Fujimoto et al., 2019b)	1.0
	VAE latent size	$2 \times \|A\|$
	VAE pretraining steps	500000
	Perturbation range	Appendix D in Zhou et al. (2020)
TD3+BC (Fujimoto and Gu, 2021)	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	256
	Policy noise	0.2
	Policy noise clipping	(-0.5, 0.5)
	Policy update frequency	2
	α (Fujimoto and Gu, 2021)	2.5
	Observation preprocess	standardization

Table 1: Hyperparameters for D4RL.

4. https://github.com/Farama-Foundation/d4rl_evaluations
5. https://github.com/aviralkumar2907/CQL

Dataset	SAC	AWAC	BCQ	BEAR	CQL	CRR	IQL	PLAS	PLAS+P	TD3+BC
halfcheetah-random-v0	30.2 ± 1.9	15.2 ± 1.3	2.3 ± 0.0	2.3 ± 0.0	29.7 ± 1.5	18.9 ± 7.0	14.4 ± 2.3	26.9 ± 1.4	27.3 ± 2.3	11.4 ± 1.5
walker2d-random-v0	2.5 ± 1.7	4.3 ± 1.9	4.2 ± 1.5	4.6 ± 1.4	2.0 ± 2.8	2.3 ± 1.9	5.8 ± 0.3	6.5 ± 7.2	4.5 ± 5.0	2.1 ± 2.4
hopper-random-v0	1.1 ± 0.6	11.1 ± 0.1	10.5 ± 0.2	10.1 ± 0.2	10.8 ± 0.1	10.9 ± 1.3	11.1 ± 0.1	10.4 ± 0.3	12.5 ± 1.5	10.9 ± 0.1
halfcheetah-medium-v0	30.1 ± 10.6	41.9 ± 0.4	40.1 ± 0.5	36.6 ± 0.9	41.7 ± 0.2	41.9 ± 0.4	41.1 ± 0.2	39.8 ± 0.4	42.0 ± 0.6	42.4 ± 1.5
walker2d-medium-v0	2.1 ± 5.3	65.6 ± 11.3	47.7 ± 7.8	57.4 ± 10.5	77.8 ± 5.1	43.3 ± 17.5	59.7 ± 8.9	33.0 ± 9.7	66.1 ± 6.8	76.7 ± 3.2
hopper-medium-v0	1.2 ± 0.8	40.7 ± 20.3	52.5 ± 22.8	34.8 ± 8.0	50.1 ± 19.9	26.1 ± 32.3	31.1 ± 0.3	58.1 ± 23.7	53.6 ± 24.0	95.9 ± 12.1
halfcheetah-medium-replay-v0	38.1 ± 7.3	42.4 ± 1.4	39.0 ± 1.9	37.1 ± 2.0	43.6 ± 3.0	42.0 ± 0.4	40.9 ± 0.9	43.9 ± 0.4	44.8 ± 1.0	42.8 ± 2.1
walker2d-medium-replay-v0	4.5 ± 3.8	22.5 ± 7.0	15.2 ± 4.6	13.6 ± 3.1	20.5 ± 4.2	26.7 ± 4.6	14.3 ± 4.4	20.9 ± 13.3	9.2 ± 10.4	26.4 ± 5.7
hopper-medium-replay-v0	8.5 ± 11.8	34.0 ± 4.1	16.1 ± 7.7	27.8 ± 6.3	31.1 ± 3.4	13.3 ± 14.0	38.1 ± 5.3	17.5 ± 13.3	20.0 ± 26.2	31.4 ± 8.0
halfcheetah-medium-expert-v0	0.0 ± 2.3	16.1 ± 4.2	60.1 ± 11.1	45.4 ± 7.4	9.0 ± 2.8	8.4 ± 1.7	55.2 ± 7.9	81.2 ± 9.6	81.6 ± 8.1	89.2 ± 6.8
walker2d-medium-expert-v0	1.7 ± 2.9	7.9 ± 22.0	43.6 ± 14.0	59.1 ± 9.6	54.7 ± 37.3	41.1 ± 12.3	92.3 ± 22.3	93.5 ± 9.1	86.3 ± 10.4	91.0 ± 14.4
hopper-medium-expert-v0	7.8 ± 9.1	42.7 ± 46.5	111.5 ± 2.8	77.8 ± 19.9	105.1 ± 7.8	0.8 ± 0.1	112.3 ± 0.2	110.8 ± 31.8	77.5 ± 50.4	112.3 ± 0.3

Table 2: Normalized scores and standard deviation collected with the final trained policy in D4RL. The scores and are averaged over 10 random seeds.

Dataset	d3rlpy	reference
halfcheetah-random-v0	30.2 ± 1.9	30.5
walker2d-random-v0	2.5 ± 1.7	4.1
hopper-random-v0	1.1 ± 0.6	11.3
halfcheetah-medium-v0	30.1 ± 10.6	-4.3
walker2d-medium-v0	2.1 ± 5.3	0.9
hopper-medium-v0	1.2 ± 0.8	0.8
halfcheetah-medium-replay-v0	38.1 ± 7.3	-2.4
walker2d-medium-replay-v0	4.5 ± 3.8	1.9
hopper-medium-replay-v0	8.5 ± 11.8	3.5
halfcheetah-medium-expert-v0	0.0 ± 2.3	1.8
walker2d-medium-expert-v0	1.7 ± 2.9	-0.1
hopper-medium-expert-v0	7.8 ± 9.1	1.6

Table 3: Side-by-side comparison with reference normalized scores of SAC reported in Fu et al. (2020), which only provides mean scores.

Dataset	d3rlpy	reference
halfcheetah-random-v0	15.2 ± 1.3	2.2
walker2d-random-v0	4.3 ± 1.9	5.1
hopper-random-v0	11.1 ± 0.1	9.6
halfcheetah-medium-v0	41.9 ± 0.4	37.4
walker2d-medium-v0	65.6 ± 11.3	30.1
hopper-medium-v0	40.7 ± 20.3	72.0
halfcheetah-medium-replay-v0	42.4 ± 1.4	-
walker2d-medium-replay-v0	22.5 ± 7.0	-
hopper-medium-replay-v0	34.0 ± 4.1	-
halfcheetah-medium-expert-v0	16.1 ± 4.2	36.8
walker2d-medium-expert-v0	7.9 ± 22.0	42.7
hopper-medium-expert-v0	42.7 ± 46.5	80.9

Table 4: Side-by-side comparison with reference normalized scores of AWAC reported in Nair et al. (2020), which only provides mean scores.

6. `https://github.com/ikostrikov/implicit_q_learning`

Dataset	d3rlpy	reference
halfcheetah-random-v0	2.3 ± 0.0	2.2
walker2d-random-v0	4.2 ± 1.5	4.9
hopper-random-v0	10.5 ± 0.2	10.6
halfcheetah-medium-v0	40.1 ± 0.5	40.7
walker2d-medium-v0	47.7 ± 7.8	53.1
hopper-medium-v0	52.5 ± 22.8	54.5
halfcheetah-medium-replay-v0	16.1 ± 7.7	38.2
walker2d-medium-replay-v0	15.2 ± 4.6	15.0
hopper-medium-replay-v0	16.1 ± 7.7	33.1
halfcheetah-medium-expert-v0	60.1 ± 11.1	64.7
walker2d-medium-expert-v0	43.6 ± 14.0	57.5
hopper-medium-expert-v0	111.5 ± 2.8	110.9

Table 5: Side-by-side comparison with reference normalized scores of BCQ reported in Fu et al. (2020), which only provides mean scores.

Dataset	d3rlpy	reference
halfcheetah-random-v0	2.3 ± 0.0	2.3 ± 2.3
walker2d-random-v0	4.6 ± 1.4	9.0 ± 6.2
hopper-random-v0	10.1 ± 0.2	10.0 ± 0.7
halfcheetah-medium-v0	36.6 ± 0.9	37.1 ± 2.3
walker2d-medium-v0	57.4 ± 10.5	56.1 ± 8.5
hopper-medium-v0	34.8 ± 8.0	30.8 ± 0.9
halfcheetah-medium-replay-v0	37.1 ± 2.0	36.2 ± 5.6
walker2d-medium-replay-v0	13.6 ± 3.1	13.7 ± 2.1
hopper-medium-replay-v0	27.8 ± 6.3	31.1 ± 0.9
halfcheetah-medium-expert-v0	45.4 ± 7.4	44.2 ± 13.8
walker2d-medium-expert-v0	59.1 ± 9.6	43.8 ± 6.0
hopper-medium-expert-v0	77.8 ± 19.9	67.3 ± 32.5

Table 6: Side-by-side comparison with reference normalized scores of BEAR collected by executing the author-provided implementation to match the hyperparameters suggested in their GitHub page.

Dataset	d3rlpy	reference
halfcheetah-random-v0	29.7 ± 1.5	28.5 ± 2.4
walker2d-random-v0	2.0 ± 2.8	1.2 ± 2.6
hopper-random-v0	10.8 ± 0.1	10.6 ± 0.8
halfcheetah-medium-v0	41.7 ± 0.2	38.8 ± 2.5
walker2d-medium-v0	77.8 ± 5.1	48.7 ± 22.2
hopper-medium-v0	50.1 ± 19.9	31.2 ± 1.0
halfcheetah-medium-replay-v0	43.6 ± 3.0	44.9 ± 2.8
walker2d-medium-replay-v0	20.5 ± 4.2	25.5 ± 13.0
hopper-medium-replay-v0	31.1 ± 3.4	30.1 ± 2.2
halfcheetah-medium-expert-v0	9.0 ± 2.8	11.3 ± 4.9
walker2d-medium-expert-v0	54.7 ± 37.3	75.4 ± 52.8
hopper-medium-expert-v0	105.1 ± 7.8	100.0 ± 18.6

Table 7: Side-by-side comparison with reference normalized scores of CQL collected by executing the author-provided implementation to match the hyperparameters suggested in their GitHub page.

Dataset	d3rlpy	reference
halfcheetah-random-v0	14.4 ± 2.3	12.2 ± 3.4
walker2d-random-v0	5.8 ± 0.3	5.7 ± 0.1
hopper-random-v0	11.1 ± 0.1	11.3 ± 0.1
halfcheetah-medium-v0	41.1 ± 0.2	41.0 ± 0.4
walker2d-medium-v0	59.7 ± 8.9	62.8 ± 5.5
hopper-medium-v0	31.1 ± 0.3	31.6 ± 0.3
halfcheetah-medium-replay-v0	40.9 ± 0.9	39.2 ± 1.6
walker2d-medium-replay-v0	14.3 ± 4.4	15.3 ± 2.4
hopper-medium-replay-v0	38.1 ± 5.3	39.1 ± 2.7
halfcheetah-medium-expert-v0	55.2 ± 7.9	54.3 ± 2.2
walker2d-medium-expert-v0	92.3 ± 22.3	101.1 ± 7.4
hopper-medium-expert-v0	112.3 ± 0.2	100.5 ± 16.8

Table 8: Side-by-side comparison with reference normalized scores of IQL collected by executing the author-provided[6] implementation because the scores for -v0 datasets were not reported in their original paper (Kostrikov et al., 2021).

Dataset	d3rlpy	reference
halfcheetah-random-v0	26.9 ± 1.4	25.8
walker2d-random-v0	6.5 ± 7.2	3.1
hopper-random-v0	10.4 ± 0.3	10.5
halfcheetah-medium-v0	39.8 ± 0.4	39.3
walker2d-medium-v0	33.0 ± 9.7	44.6
hopper-medium-v0	58.1 ± 23.7	32.9
halfcheetah-medium-replay-v0	38.1 ± 7.3	43.9
walker2d-medium-replay-v0	20.9 ± 13.3	30.2
hopper-medium-replay-v0	17.5 ± 13.3	27.9
halfcheetah-medium-expert-v0	81.2 ± 9.6	96.6
walker2d-medium-expert-v0	93.5 ± 9.1	89.6
hopper-medium-expert-v0	100.8 ± 31.8	110.0

Table 9: Side-by-side comparison with reference normalized scores of PLAS reported in Zhou et al. (2020), which only provides mean scores.

Dataset	d3rlpy	reference
halfcheetah-random-v0	27.3 ± 2.3	28.3
walker2d-random-v0	4.5 ± 5.0	6.8
hopper-random-v0	12.5 ± 1.5	13.3
halfcheetah-medium-v0	42.0 ± 0.6	42.2
walker2d-medium-v0	66.1 ± 6.8	66.9
hopper-medium-v0	53.6 ± 24.0	36.9
halfcheetah-medium-replay-v0	44.8 ± 1.0	45.7
walker2d-medium-replay-v0	9.2 ± 10.4	14.3
hopper-medium-replay-v0	20.0 ± 26.2	51.9
halfcheetah-medium-expert-v0	81.6 ± 8.1	99.3
walker2d-medium-expert-v0	86.3 ± 10.4	96.2
hopper-medium-expert-v0	77.5 ± 50.4	94.7

Table 10: Side-by-side comparison with reference normalized scores of PLAS+P reported in Zhou et al. (2020), which only provides mean scores.

Dataset	d3rlpy	reference
halfcheetah-random-v0	11.4 ± 1.5	10.2 ± 1.3
walker2d-random-v0	2.1 ± 2.5	1.4 ± 1.6
hopper-random-v0	10.9 ± 0.1	11.0 ± 0.1
halfcheetah-medium-v0	42.4 ± 0.5	42.8 ± 0.3
walker2d-medium-v0	76.7 ± 3.2	79.7 ± 1.8
hopper-medium-v0	95.9 ± 12.1	99.5 ± 1.0
halfcheetah-medium-replay-v0	42.8 ± 2.1	43.3 ± 0.5
walker2d-medium-replay-v0	26.4 ± 5.7	25.2 ± 5.1
hopper-medium-replay-v0	31.4 ± 8.0	31.4 ± 3.0
halfcheetah-medium-expert-v0	89.2 ± 6.8	97.9 ± 4.4
walker2d-medium-expert-v0	91.0 ± 14.4	101.1 ± 9.3
hopper-medium-expert-v0	112.3 ± 0.3	112.2 ± 0.2

Table 11: Side-by-side comparison with reference normalized scores of TD3+BC reported in Fujimoto and Gu (2021), which provides standard deviations as well as mean scores.

Appendix B. Benchmark Results: Atari 2600

We evaluated the implemented algorithms with open-sourced Atari 2600 datasets (Agarwal et al., 2020). We followed the experimental procedure described in Agarwal et al. (2020). We used 1% portion of transitions (500K datapoints) and train each algorithm for 12.5M gradient steps and evaluate every 125K steps to collect evaluation performance in environments for 10 episodes. Table 12 shows hyperparameters used in benchmarking. We used the same hyperparameters for QR-DQN and CQL as the ones reported in Kumar et al. (2020). For NFQ and BCQ, the hyperparameters were chosen based on the QR-DQN setup for fair comparison because there are no benchmark results for them with publically available datasets. We used discount factor of 0.99, Adam optimizer (Kingma and Ba, 2014) and the convolutional neural network (Mnih et al., 2015) across all algorithms. Note that we configured BCQ with the Quantile Regression Q-function introduced in Section 3.2 to match the CQL setup. In evaluation, we used ϵ-greedy of $\epsilon = 0.001$ and 25% probability of sticky action (Agarwal et al., 2020). We repeated all experiments with 10 random seeds.

Table 13 shows the benchmark results, and Table 14 and 15 show side-by-side comparisons with reference scores. NFQ and BCQ are not included in the side-by-side comparisons because those are not evaluated with publically available datasets, an author-provided implementation for NFQ is not publically available, and the author-provided implementation[7] for BCQ is not directly applicable for this evaluation. Considering that the authors did not report standard deviations, we believe that performance discrepancy is trivial.

Algorithm	Hyperparameter	Value
NFQ	Learning rate	5e-5
	Mini-batch size	32
QR-DQN	Learning rate	5e-5
	Mini-batch size	32
	ϵ (Kingma and Ba, 2014)	3.125e-4
	Number of quantiles	200
	Target update frequency	2000
BCQ	Learning rate	5e-5
	Mini-batch size	32
	ϵ (Kingma and Ba, 2014)	3.125e-4
	Number of quantiles	200
	Target update frequency	2000
	τ (Fujimoto et al., 2019a)	0.3
	Pre-activation regularization (Fujimoto et al., 2019a)	1e-2
CQL	Learning rate	5e-5
	Mini-batch size	32
	ϵ (Kingma and Ba, 2014)	3.125e-4
	Number of quantiles	200
	Target update frequency	2000
	α (Kumar et al., 2020)	4.0

Table 12: Hyperparameters for Atari 2600 datasets.

7. `https://github.com/sfujim/BCQ`

Dataset	**NFQ**	**QR-DQN**	**BCQ**	**CQL**
Pong	-18.0 ± 0.9	-16.6 ± 1.1	13.8 ± 1.5	16.3 ± 1.4
Breakout	6.4 ± 1.0	9.0 ± 1.9	40.7 ± 10.8	89.8 ± 12.2
Qbert	648.5 ± 81.4	695.8 ± 153.1	8058.8 ± 926.1	$15\,791.5 \pm 530.5$
Seaquest	841.2 ± 67.3	603.2 ± 62.3	1005.1 ± 175.5	820.5 ± 94.0
Asterix	745.0 ± 85.6	551.0 ± 76.2	857.0 ± 61.2	1636.5 ± 166.5

Table 13: The raw scores collected with the best trained policy in 1% portion of Atari 2600 datasets. The scores are averaged over 10 random seeds.

Dataset	d3rlpy	reference
Pong	-16.6 ± 1.1	-13.8
Breakout	9.0 ± 1.9	7.9
Qbert	695.8 ± 153.1	383.6
Seaquest	603.2 ± 62.3	672.9
Asterix	551.0 ± 76.2	166.3

Table 14: Side-by-side comparison with reference raw scores of QR-DQN reported in Kumar et al. (2020), which only provides mean scores.

Dataset	d3rlpy	reference
Pong	16.3 ± 1.4	19.3
Breakout	89.8 ± 12.2	61.1
Qbert	$15\,791.5 \pm 530.5$	14012.0
Seaquest	820.5 ± 94.0	779.4
Asterix	1636.5 ± 166.5	592.4

Table 15: Side-by-side comparison with reference raw scores of CQL reported in Kumar et al. (2020), which only provides mean scores.

Appendix C. Benchmark Results: Fine-tuning

We evaluated the implemented algorithms with AntMaze datasets (Fu et al., 2020) in a fine-tuning scenario where a policy is pretrained with a static dataset and fine-tuned with online experiences. We followed the experimental procedure described in Kostrikov et al. (2021). In this evaluation, we chose AWAC and IQL, which are proposed as RL algorithms with fine-tuning capability. Table 16 shows hyperparameters used in benchmarking. All reward values are subtracted by 1. We used the same hyperparameters described in the original paper (Nair et al., 2020; Kostrikov et al., 2021). In each training, the policy was fine-tuned for 1M steps after pretraining. We repeated all experiments with 10 random seeds.

Table 17 shows benchmark results. Table 18 and 19 show side-by-side comparisons with reference scores. Considering that the authors did not report standard deviations, we believe that performance discrepancy is trivial.

Algorithm	Hyperparameter	Value
AWAC (Nair et al., 2020)	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	1024
	λ (Nair et al., 2020)	1
	Actor hidden units	[256, 256, 256, 256]
	Actor weight decay	1e-4
	Critic hidden units	[256, 256, 256, 256]
	Pretraining steps	25000
IQL (Kostrikov et al., 2021)	Critic learning rate	3e-4
	Actor learning rate	3e-4
	V-function learning rate	3e-4
	Mini-batch size	256
	Expectile	0.9
	Inverse temperature	10.0
	Actor learning rate scheduler	Cosine
	Pretraining steps	1M

Table 16: Hyperparameters for fine-tuning experiments.

Dataset	**AWAC**	**IQL**
antmaze-umaze-v0	52.0±24.4 → 92.0±20.9	92.0±4.0 → 98.0±4.0
antmaze-medium-play-v0	0.0±0.0 → 0.0±0.0	71.0±14.5 → 91.0±9.4
antmaze-large-play-v0	0.0±0.0 → 0.0±0.0	52.0±16.6 → 69.0±13.7

Table 17: The normalized scores collected with the final trained policy in AntMaze datasets (Fu et al., 2020). The numbers on the left represent scores of the pretrained policies. The numbers on the right represent scores of the fine-tuned policies. The scores are averaged over 10 random seeds.

Dataset	**d3rlpy**	**reference**
antmaze-umaze-v0	52.0±24.4 → 92.0±20.9	56.7 → 59.0
antmaze-medium-play-v0	0.0±0.0 → 0.0±0.0	0.0 → 0.0
antmaze-large-play-v0	0.0±0.0 → 0.0±0.0	0.0 → 0.0

Table 18: Side-by-side comparison with reference normalized scores of AWAC reported in Kostrikov et al. (2021), which only provides mean scores.

Dataset	**d3rlpy**	**reference**
antmaze-umaze-v0	92.0±4.0 → 98.0±4.0	86.7 → 96.0
antmaze-medium-play-v0	71.0±14.5 → 91.0±9.4	72.0 → 95.0
antmaze-large-play-v0	52.0±16.6 → 69.0±13.7	25.5 → 46.0

Table 19: Side-by-side comparison with reference normalized scores of IQL reported in Kostrikov et al. (2021), which only provides mean scores.

References

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL `http://arxiv.org/abs/1812.06110`.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018a.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018b.

Ludovic Denoyer, Alfredo de la Fuente, Song Duong, Jean-Baptiste Gaya, Pierre-Alexandre Kamienny, and Daniel H Thompson. Salina: Sequential learning of agents. *arXiv preprint arXiv:2110.07910*, 2021.

Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Mushroomrl: Simplifying reinforcement learning research. 2021.

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.

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

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1587–1596, 2018.

Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019a.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019b.

Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. Chainerrl: A deep reinforcement learning library. *Journal of Machine Learning Research*, 22(77):1–14, 2021. URL `http://jmlr.org/papers/v22/20-376.html`.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

P Henderson, R Islam, P Bachman, J Pineau, D Precup, and D Meger. Deep reinforcement learning that matters. arxiv 2017. *arXiv preprint arXiv:1709.06560*, 2017.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

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

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. Web page, 2017. URL `https://github.com/tensorforce/tensorforce`.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

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

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

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.

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. In *International Conference on Learning Representation*, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Fabio Pardo. Tonic: A deep reinforcement learning library for fast prototyping and benchmarking. *arXiv preprint arXiv:2011.07537*, 2020.

Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. In *International Conference on Machine Learning*, pages 4045–4054. PMLR, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12: 2825–2830, 2011.

Matthias Plappert. keras-rl. `https://github.com/keras-rl/keras-rl`, 2016.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, pages 317–328. Springer, 2005.

Rutav Shah and Vikash Kumar. Rrl: Resnet as representation for reinforcement learning. *arXiv preprint arXiv:2107.03380*, 2021.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. corr abs/1509.06461 (2015). *arXiv preprint arXiv:1509.06461*, 2015.

Ziyu Wang, Alexander Novikov, Konrad Żołna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *arXiv preprint arXiv:2006.15134*, 2020.

Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Hang Su, and Jun Zhu. Tianshou: A highly modularized deep reinforcement learning library. *arXiv preprint arXiv:2107.14171*, 2021.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

Wenxuan Zhou, Sujay Bajracharya, and David Held. Latent action space for offline reinforcement learning. In *Conference on Robot Learning*, 2020.