Evaluation of Sample Efficiency in Offline Reinforcement Learning

by

Shivakanth SUJIT

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN INFORMATION TECHNOLOGY ENGINEERING
M.A.Sc.

MONTREAL, DECEMBER 6TH 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mrs. Samira Ebrahimi Kahou, Thesis supervisor
Département de génie logiciel et des technologies de l'information,
École de Technologie Supérieure

Mr. Sheldon Andrews, Chair, Board of Examiners
Département de génie logiciel et des technologies de l'information,
École de Technologie Supérieure

Mr. Adrien Gruson, Member of the Jury
Département de génie logiciel et des technologies de l'information,
École de Technologie Supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON NOVEMBER 13TH 2023

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I am grateful for all the support I have received during my graduate program. I would like to thank Prof. Samira Ebrahimi Kahou for her invaluable guidance and help throughout the program. She was an instrumental part of my introduction to research and has pushed me to become better with every project.

I want to thank my collaborators who have made the research journey enjoyable and productive: Arnav Jain, Somjit Nath, Pedro Braga, Ivaxi Sheth, Vincent Michalski, Danijar Hafner and Jorg Bornschein. A graduate program can be a long, arduous process, and good friends inject levity to these moments, whether through long conversations about machine learning research or banal everyday topics. For that I am forever grateful to Kargil Mishra, Jith Subramanian, Soma Karthik, Nanda Krishna, Kshitij Gupta, Moksh Jain, Aniket Didolkar, Nishka Katoch, Prishruit Punia, Shruti Joshi, Mohammad Reza Samsami, Rohan Banerjee and Anirudh Kemtur. I would also like to thank Max Schwarzer, Pierluca D'Oro, Evgenii Nikishin, Jacob Buckman, Moustafa Elarabi for their invaluable feedback and advice throughout my program. I am grateful for all the support my parents have given, instilling in me a sense of wonder and curiosity about the world that continues to drive my research.

Finally I would be remiss if I did not express my gratitude to Digital Research Alliance of Canada for compute resources, and NSERC, Google and CIFAR for research funding, all of which have made my research possible.

Évaluation de l'efficacité des données dans l'apprentissage par renforcement hors ligne

Shivakanth SUJIT

RÉSUMÉ

L'apprentissage par renforcement (RL) s'est avéré très prometteur avec des algorithmes apprenant dans des environnements avec de grands espaces d'état et d'action uniquement à partir de signaux de récompense scalaires. L'un des principaux défis des algorithmes actuels d'apprentissage par renforcement est qu'ils nécessitent un nombre considérable d'interactions avec l'environnement pour l'apprentissage. Cela peut s'avérer infaisable dans les situations où ces interactions sont coûteuses, comme en robotique. Les algorithmes RL hors ligne tentent de résoudre ce problème en amorçant le processus d'apprentissage à partir des données enregistrées existantes sans avoir besoin d'interagir avec l'environnement dès le départ. Alors que les algorithmes RL en ligne sont généralement évalués en fonction du nombre d'interactions avec l'environnement, il n'existe pas de protocole unique établi pour évaluer les méthodes RL hors ligne. Dans cette thèse, nous proposons une approche séquentielle pour évaluer les algorithmes RL hors ligne en fonction de la taille de l'ensemble d'apprentissage et donc de leur efficacité en termes de données. L'évaluation séquentielle fournit des informations précieuses sur l'efficacité du processus d'apprentissage et la robustesse des algorithmes aux changements de distribution dans l'ensemble de données, tout en harmonisant la visualisation des phases d'apprentissage en ligne et hors ligne. Notre approche est généralement applicable et facile à mettre en œuvre. Nous comparons plusieurs algorithmes RL hors ligne existants à l'aide de cette approche et présentons les résultats d'une variété de tâches et d'ensembles de données hors ligne.

Mots-clés: apprentissage par renforcement hors ligne, méthodes d'évaluation

Evaluation of Sample Efficiency in Offline Reinforcement Learning

Shivakanth SUJIT

ABSTRACT

Reinforcement learning (RL) has shown great promise with algorithms learning in environments with large state and action spaces purely from scalar reward signals. A crucial challenge for current deep RL algorithms is that they require a tremendous amount of environment interactions for learning. This can be infeasible in situations where such interactions are expensive; such as in robotics. Offline RL algorithms try to address this issue by bootstrapping the learning process from existing logged data without needing to interact with the environment from the very beginning. While online RL algorithms are typically evaluated as a function of the number of environment interactions, there exists no single established protocol for evaluating offline RL methods. In this thesis, we propose a sequential approach to evaluate offline RL algorithms as a function of the training set size and thus by their data efficiency. Sequential evaluation provides valuable insights into the data efficiency of the learning process and the robustness of algorithms to distribution changes in the dataset while also harmonizing the visualization of the offline and online learning phases. Our approach is generally applicable and easy to implement. We compare several existing offline RL algorithms using this approach and present insights from a variety of tasks and offline datasets.

Keywords: offline reinforcement learning, evaluation methods

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

ETS	École de Technologie Supérieure
RL	Reinforcement learning
DQN	Deep Q Network
MDP	Markov Decision Process
TD	temporal difference
IS	importance sampling
TRPO	Trust Region Policy Optimization
PPO	Proximal Policy Optimization
BC	Behavior cloning
RWR	Reward Weighted Regression
CRR	Critic Regularized Regression
AWR	Advantage Weighted Regression
OOD	out of distribution
CQL	Conservative Q Learning
IQL	Implicit Q Learning
BCQ	Batch Constrained Q Learning
DT	Decision Transformer
SeqEval	sequential evaluation
ERM	empirical risk minimization

DL	Deep Learning
MDL	Minimum Description Length
DMC	DeepMind Control Suite
SAC	Soft Actor Critic
IQM	interquartile mean

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

π	Agent policy
Q_π	Q value of the policy π
V_θ	State value of the policy π
$\mathcal{S}$	State space of the environment
$\mathcal{A}$	Action space of the environment
$\mathcal{R}$	Reward function of the environment
$\mathcal{P}$	Transition function of the environment
γ	Discount factor
s_t	State from the environment at timestep t
a_t	Action executed by the agent at timestep t
r_t	Reward obtained at timestep t
G_t	Return achieved by an agent in an episode

INTRODUCTION

Reinforcement learning (RL) is a paradigm for learning when well defined ground truth labels or behavior are not available for training. In the absence of an explicit ground truth label for supervision, the agent learns from trial and error, where it is penalised for behavior leading to bad outcomes and rewarded for behavior that leads to good outcomes. Historically, reinforcement learning has shown promise in learning agents that can play simple games, such as backgammon (Tesauro, 1995) or checkers (Samuel, 1959). But such successes were marred by the difficulty in extending them to play harder games with complex dynamics and interactions. These approaches were computationally intensive and did not scale to games with high dimensional inputs such as images. However the rise in the use of deep neural networks for image classification (Krizhevsky, Sutskever & Hinton, 2012), object detection (Girshick, Donahue, Darrell & Malik, 2013) and segmentation (Shelhamer, Long & Darrell, 2014) tasks spurred interest in combining these expressive deep networks with RL. This led to a major breakthrough by Mnih *et al.* (2013), where a deep neural network was trained to play Atari games directly from images and reward signals from the game. The algorithm, named Deep Q Network (DQN) learnt to play seven Atari games and even achieved human-level performance on three of the games.

The above mentioned approaches used online reinforcement learning for solving these tasks. Online reinforcement learning assumes that the agent has access to an environment (either in the real world or a simulator) that it can interact with. This interaction serves two purposes: 1) it allows the agent to collect more data to be trained on, and 2) it allows the agent to validate its current beliefs about the world. The former is desirable since deep learning methods tend to perform better with access to more data (Kaplan *et al.*, 2020), and having a ready source of new data assists in this purpose. It can be said that the latter objective is a by-product of the former, but there is a subtle difference. Online RL differs from other forms of deep learning in that the model has a role to play in the data collection process. The behavior of the model actively affects the distribution of data that it collects. This is not the case in computer vision or

natural language tasks in deep learning where the model plays no part in the data collection process. This model dependence in the data collection process makes RL challenging since a poor policy can be doomed to remain at sub-optimal behavior since it might not ever collect data representing optimal behavior, thereby preventing learning such behavior. Interacting in the environment serves as a kind of grounding whereby the agent can execute behavior that it believes leads to good outcomes and observe if that is actually the case. This ability to test the quality of its behavior has been shown to be essential for stable learning (Ostrovski, Castro & Dabney, 2021; Fujimoto, van Hoof & Meger, 2018). In Ostrovski *et al.* (2021), the authors conduct an experiment with two RL agents that train on the same data, but the data is collected by only one agent. That is, only one agent is allowed to interact with the environment and collect data, while the other agent can only passively learn from this data. They control for the sampling of mini-batches from the buffer to ensure that both agents train on the exact same data and only differ in the initialization of the neural network weights. Their experiments show that while the online agent is able to improve and solve the task, the passive learner is not able to solve the task. Even though the agents are trained with exactly the same data, the passive agent does not learn successfully. They attribute this behavior to the inability of the passive learner to correct its mistakes. If the online agent overestimated the goodness of its actions, it would receive corrective feedback from the environment in the form of lower rewards than expected. The passive learner, on the other hand, does not receive such feedback. If it erroneously believes that a given action is optimal, it has no ability to correct this mistake. Hence it would continue to have these beliefs and perform poorly.

This result highlights the difficulty in learning purely from a fixed dataset whose collection the agent can not influence and the branch of RL that studies this setting is termed offline RL. Offline RL methods need to create a balance between mimicking the action distribution in the dataset and generalizing outside of the exact states observed in the dataset. For example, if the learnt policy overfits to the dataset, it can perform poorly when the environment deviates slightly

from previously seen behavior. Similarly we have to be careful about how the agent extrapolates outside of the dataset since it wouldn't have the ability to check if these extrapolations hold up in the actual environment.

In this thesis, we do not study algorithm design in offline RL, but focus on devising new evaluation protocols to test for sample efficiency in offline RL. The traditional approach to evaluation has been what we term the "mini-batch" style of evaluation, where algorithms are tested as a function of gradient steps. The agent is given access to the entire dataset from which it can sample minibatches of data at a time to train on. The algorithms are given a fixed compute budget, usually around 1M gradient steps, and performance is evaluated periodically throughout training. The performance curves reported are therefore a function of the compute afforded to the algorithm. But this style of evaluation does not provide any insights into the data scaling properties of the algorithm. It is not possible to answer questions pertaining to how the performance would vary if you doubled the amount of data that the algorithm had access to, or halved it. These properties are useful to have when performing algorithm selection for practical applications since final performance might not be the only concern. There could be a bottleneck in the amount of data available for the task and knowing how the algorithm scales with data could inform if it is worthwhile to spend extra time collecting more data. Furthermore, mini-batch style training does not study how the algorithm reacts to changes in data distribution since it is trained on all data at the same time. But a dataset could have been collected by policies of varying quality and it can be useful to know if a given algorithm can adapt to new data and learn from it, or if it would be better to just train from scratch.

To address these concerns, we study a sequential style of evaluation of offline RL algorithms. Sequential evaluation varies the amount of data available to an algorithm during training and studies how they improve with dataset sizes and changes in data distribution. We provide a concrete implementation of this approach and showcase how it can be utilized for performing

algorithm selection in offline RL. We show this across varied datasets and algorithms to highlight the generality of the sequential approach. Finally, we provide recommendations for how algorithm designers can showcase the results of sequential evaluation in through model cards summarizing key data scaling properties.

The rest of this thesis is organized as follows. Chapter 1 provides an overview of RL formalisms and definitions, followed by a discussion of online and offline RL methods and challenges present in training these methods. Chapter 2 describes the organization of the articles in the thesis. Chapter 3 presents the sequential evaluation framework, main experimental results and design recommendations. Chapter 4 contextualizes the results of the article with respect to the rest of the thesis and finally, Conclusion and Recommendations summarizes our work and the main takeaways that we wish to leave the reader with. Appendix I contains additional results and descriptions of the datasets and algorithms used.

CHAPTER 1

LITERATURE REVIEW

1.1 Preliminaries

Reinforcement learning (RL) is concerned with learning when ground truth supervision targets are not available, but the fitness/quality of an action/prediction can be evaluated. In this situation, the agent learns through trial and error, initially following a random policy, but over time learning a policy that maximises the "goodness" of its actions.

The RL paradigm follows the structure as given in Fig. 1.1. An agent is assumed to be interacting in an environment, at each timestep receiving a state s_t from the environment. Based on the state, it chooses an action a_t to execute in the environment. The execution of an action a_t causes the environment to transition to the next state s_{t+1} and the agent receives a reward r_t. The reward is a measure of how good the action a_t is. The environment is modelled as a Markov Decision Process (MDP) (Bellman, 1957), described by the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} >$. $\mathcal{S}$ corresponds to the set of possible states in the environment, $\mathcal{A}$ is the set of valid actions, $\mathcal{R} : \mathcal{S} \times \mathcal{A}$ is the reward function, and $\mathcal{P}$ defines the transition dynamics of the environment.

The environment can follow an episodic setting or an infinite horizon setting. In the episodic setting, the environment will terminate after a finite number of timesteps. In the infinite horizon setting, there can be situations where an episode does not terminate at all. We define the discounted return as $G = \sum_{t=0}^{T} \gamma^t r_t$. The discount factor $\gamma \in [0, 1]$ specifies the tradeoff between recent rewards and future rewards. A discount factor of 1 means all rewards are treated equally, whether the rewards were obtained in the first timestep or the thousandth timestep. This can be problematic in the infinite horizon setting since the sum of rewards will diverge. Furthermore, we would like to encourage immediate rewards over future rewards and this can be done by setting $\gamma \in [0, 1)$. This means that a reward r_{t+k} is less important than r_t by a factor of γ^k. It also ensures that the return is bounded in infinite-length episodes. The closer γ is to 0, the more the agent will value greedy immediate rewards over long term rewards.

Figure 1.1 Visualization of the Agent-Environment loop in reinforcement learning.

The agent in RL is modelled through a policy $\pi(a_t|s_t)$ that provides a mapping from states to actions. This mapping can be modelled implicitly or explicitly as we will see in value-based and policy gradient methods. There is no restriction on the type of functions that can be used to represent this mapping, though in recent years there has been great success in using neural network based function approximators (Mnih *et al.*, 2013; Schulman, Wolski, Dhariwal, Radford & Klimov, 2017; Bellemare, Dabney & Munos, 2017; Berner *et al.*, 2019). The objective in RL is to learn a policy that maximises the discounted return G. This policy is sometimes referred to as the *optimal* policy π^*. The learning objective in RL can hence be summarized as finding π^* where

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{a\sim\pi, s'\sim\mathcal{P}} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s'_t) \right]. \tag{1.1}$$

1.1.1 Value Based Methods

The value $V_\pi(s)$ of a state s is defined the as the expected discounted return G_t that can be obtained from a state s by following policy π. That is,

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=t'}^{T} \gamma^t \, R(s_t, a_t, s'_t) \right] \tag{1.2}$$

$$= \mathbb{E}_\pi \left[G_t | s_t = s \right] . \tag{1.3}$$

Intuitively, the value of a state is a measure of the expected "goodness" of a state. Similarly, the Q value, $Q_\pi(s_t, a_t)$ of a state, action pair is the expected return from taking action a_t in state s_t and thereafter following policy π to select actions. That is,

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]. \tag{1.4}$$

By inspecting Eqs. 1.2 and 1.4, we see that the Q value can be defined in terms of the value of a state, $Q_\pi(s, a) = \mathbb{E}_{s' \sim \mathcal{P}}[r_t + \gamma V_\pi(s')]$. Additionally, the advantage of a state-action pair $A_\pi(s, a)$ is the difference between the Q value, $Q_\pi(s, a)$ and the state value, $V_\pi(s)$. The advantage $A_\pi(s, a)$ encodes a notion of how much better a given action a is compared to other actions you can take in a state s.

Value based methods model a policy implicitly by learning Q_π and using it to take the action that has the maximum Q value. That is $\pi(a|s) = \underset{a}{\text{argmax}} Q_\pi(s, a)$. The value and Q value of the optimal policy π^* are denoted as V^*_π and Q^*_π respectively.

The Bellman equations define a recursive version of the value and Q value equations of a policy.

$$V_\pi(s) = \mathbb{E}_{a \sim \pi, s' \sim \mathcal{P}} \left[r + \gamma V_\pi(s') \right] , \tag{1.5}$$

$$Q_\pi(s, a) = \mathbb{E}_{s' \sim \mathcal{P}} \left[r + \gamma \mathbb{E}_{a' \sim \pi}[Q_\pi(s', a')] \right] . \tag{1.6}$$

For the optimal policy, the equations are modified and referred to as the Bellman *optimality* equations. They reflect that the optimal policy would predict actions that maximize the value V /Q value Q function and hence are given by

$$V^*_\pi(s) = \max_a \mathbb{E}_{s' \sim \mathcal{P}} \left[r + \gamma V^*_\pi(s') \right], \tag{1.7}$$

$$Q^*_\pi(s,a) = \mathbb{E}_\pi \left[r + \gamma \max_a \left[Q^*_\pi(s',a') \right] \right]. \tag{1.8}$$

Value based methods utilize the Bellman optimality equation to learn an estimator of the Q value function. Specifically, they train a network (often termed the Q network Q_θ, parameterized by θ) to satisfy the self-consistency in Eq. 1.7. By definition, an estimator that fulfills the self-consistency of the optimality equation will be an estimator of the optimal Q value network. The Q network can be distilled to a policy π_θ by computing the Q values of every action in a state and then taking the action with the highest Q value. That is, $\pi(\cdot|s_t) = \underset{a}{\text{argmax}}\, Q_\theta(s_t, a)$. The Q network can be learnt by regressing its output to satisfy Eq. 1.7.

The difference between $Q_\theta(s_t, a_t)$ and $(r_t + \gamma \max_{a'} Q_\theta(s', a'))$ is termed the temporal difference (TD) error. Value methods learn Q_θ by minimizing its TD error. Since these methods learn the Q value function, they are also called Q learning methods. The loss function L_θ used for optimization via gradient descent is given by

$$L_\theta = \mathbb{E}_{s_t, a_t, r_t, s_{t+1} \sim \mathcal{D}} \left[(Q_\theta(s_t, a_t) - (r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a')))^2 \right]. \tag{1.9}$$

where $< s_t, a_t, r_t, s_{t+1} >$ are sampled from a buffer $\mathcal{D}$ of previous experience. An important benefit of the optimization process is that L_θ does not impose any constraints on where the data is collected from. That is, Q_θ can be optimized with data that π_θ did not collect, specifically the action a_t need not be produced from Q_θ. Such types of methods are termed off policy algorithms. Conversely, algorithms that can be trained only with data generated by the current version of the

algorithm are termed on policy algorithms. Off policy algorithms tend to have better sample complexity compared to on policy algorithms since they can reuse old data for optimization. When on policy algorithm trains on a batch of data, its policy changes and hence that batch can not be used again, so it is discarded and a fresh batch of data is collected from the environment.

The objective in Eq. 1.9 involves a maximization step to calculate $\max_{a'} Q_\theta(s_{t+1}, a')$. In environments with discrete action spaces it can be easy to find the maximum over valid actions. But in environments with continuous action spaces, it can be difficult since it would involve a separate optimization step to find the maximum Q value, which adds to the time and complexity of the process. Instead of performing this step, it has been proposed to train an actor network μ_ϕ that predicts the action that maximizes Q_θ. The max operation in Eq. 1.9 is then replaced with $Q_\theta(s_{t+1}, \mu_\phi(s_{t+1}))$ The actor network is trained in parallel with the Q network with the following loss function (the negative sign indicates that it maximizes this quantity):

$$L_\phi = -\mathbb{E}_{s_t \sim D}\left[Q_\theta(s_t, \mu_\phi(s_t))\right], \tag{1.10}$$

1.1.2 Policy Gradient Methods

While value based methods learn a policy implicitly by estimating the Q value function Q_θ, policy gradient methods explicitly learn a policy π_θ that maximizes the discounted return of the agent. That is, the objective of policy gradient methods is

$$J(\theta) = \mathbb{E}_{s \sim \mathcal{P}}[G_t] \tag{1.11}$$

$$= \mathbb{E}_{s \sim \mathcal{P}}[V_{\pi_\theta}(s)] \tag{1.12}$$

It is difficult to directly optimize this quantity via gradient descent since $\nabla_\theta J(\theta)$ will need the gradient with respect to both the action distribution of the policy and the state distribution of the

environment that is induced by the policy π_θ. While we can compute the gradient with respect to the policy, it is not simple to calculate the gradient with respect to the state distribution since it would require a differentiable environment. This can greatly limit the problems that can be solved as there can be several instances where we interact with an environment that we don't have complete access to.

The Policy Gradient Theorem (Sutton & Barto, 2018) derives an estimate of the gradient of the objective that is free from the derivative with respect to the state distribution and obtains $\nabla_\theta J(\theta) \propto \mathbb{E}_{\pi_\theta} [Q_\pi(s,a) \nabla_\theta \ln \pi_\theta(a|s)]$. This estimator of the gradient is unbiased but has high variance and there have been follow up proposals for reducing the variance of the estimator.

REINFORCE (Williams, 1992) uses the Monte Carlo episodic returns G_t in lieu of Q_θ to estimate the policy gradient. This works because over multiple samples, the expectation of G_t is Q_θ. Additionally, a baseline is subtracted from G_t to reduce the variance without affecting bias, with popular options being a running mean of the past episodic returns.

The policy gradient defined above is on policy since the gradient is defined with respect to the current policy parameters. As mentioned in Section 1.1.1, algorithms that directly use this version of the gradient can require more data as they need to repeatedly collect new data to train on. To improve sample complexity, importance sampling (IS) (Precup, Sutton & Singh, 2000) has been used to define an off policy version of the gradient that can reuse data or perform multiple gradient updates on the same batch of data. IS reweighs the contribution of a sample to the gradient based on how different the current policy is from the policy that collected the data. Assuming π_b is the behavior policy used to collect the data, each sample is reweighed by $\frac{\pi_\theta(a|s)}{\pi_b(a|s)}$. This correction changes the gradient equation as follows,

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{\pi_b} \left[\frac{\pi_\theta(a|s)}{\pi_b(a|s)} Q_\pi(s,a) \nabla_\theta \ln \pi_\theta(a|s) \right] \tag{1.13}$$

Notice that the expectation is now with respect to π_b instead of π_θ. IS is implemented by performing multiple mini-batch updates where the behavior policy is the policy before starting

the current step of optimization. IS can diverge during training if the current policy differs significantly from the behavior policy as the IS weight can explode or vanish. This limits the number of consecutive mini-batch gradient updates that can be performed before new data must be collected. There are two popular approaches to minimize the divergence between the current and behavior policy. Trust Region Policy Optimization (TRPO) (Schulman, Levine, Moritz, Jordan & Abbeel, 2015) enforces a KL divergence penalty on the distribution of the behavior policy and the current policy. This creates a constraint on how much the policy distribution can change with each mini-batch gradient update, that is $\mathbb{E}_{\pi_b}\left[D_{\mathrm{KL}}(\pi_b(\cdot|s)||\pi_\theta(\cdot|s)\right] \leq c$. While this constraint provides a guarantee on the improvement in the policy, it is difficult to implement it since it involves calculation of higher order gradients at every update which can be computationally expensive and unstable. Proximal Policy Optimization (PPO) (Schulman *et al.*, 2017) shows that the policy divergence can be mitigated by simply clipping the IS weight to remain in the interval $[1-\epsilon, 1+\epsilon]$, where ϵ is usually set to a small value such as 0.2. This clipping has the same effect as discouraging gradient updates that drastically change the policy, while being very simple to implement.

1.2 Offline Reinforcement Learning

Online RL assumes that the agent can learn by acting in an environment and collecting data. But there can be situations where it is undesirable or infeasible for the agent to interact with the system. This can be due to the risk associated with interacting in the real system (for example it can be dangerous in applications related to healthcare), or due to the cost if the interactions are expensive or time consuming. For this reason it is useful to design RL algorithms that can learn from offline datasets. This branch of RL is termed offline RL. They learn a policy purely from previously collected data either from expert demonstrations or even random data. This property makes offline RL particularly attractive for industrial applications where operators might be wary of allowing a sub-optimal policy to interact with the system, but there exists logged data showing what good behavior looks like.

The lack of a simulator for interaction makes it difficult for offline RL agents to correct misconceptions about the environment. For example, in online RL if the agent is overly optimistic about an action, it can execute the action in the environment and correct its estimate, which is not possible in offline RL. More generally, offline RL algorithms need to generalize to situations outside of the training data. But at the same time, the algorithm should not deviate too much from the dataset collecting policy. These two competing objectives need to be carefully balanced while designing an algorithm.

The simplest form of offline RL is Behavior cloning (BC) which focuses only on the latter objective. The BC agent is trained to mimic the actions in the dataset. The actions taken in the dataset are considered to be the ground truth labels for the environment states and the agent is trained to match these actions as closely as possible. Given a dataset $\mathcal{D}$ of N samples, $\{s_i, a_i, r_i\}_{i=0}^{N-1}$, BC learns a policy $\pi(a|s)$ with the following objective

$$\pi = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{s,a\sim\mathcal{D}} \left[\log \pi(a|s)\right]. \tag{1.14}$$

BC is particularly useful if there is sufficient quantity and quality of expert demonstrations. But these agents can be brittle if there is not enough coverage of the state space in the dataset. In this case, a small deviation from the trajectory followed by an expert can lead the BC agent to completely fail since it would be in a region of the state space it has not encountered before during training. Another observed failure mode of BC is when the agent is expected to stitch desired behavior across different suboptimal episodes (Chen *et al.*, 2021). To address this concern, there are approaches that assign a heuristic of how useful it is to mimic a particular action. This way, if there is an episode which overall achieves a poor outcome, but has portions of the trajectory which are desirable, the agent can assign high importance to those portions while disregarding the rest. The general form of the objective of such methods is

$$\pi = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{s,a,r\sim\mathcal{D}} \left[\Psi(s,a,r)\ \log \pi(a|s)\right], \tag{1.15}$$

where $\Psi(s, a, r)$ is an estimate of how desirable it is to take action a in state s and can be derived directly from the dataset without any learning. For example Reward Weighted Regression (RWR) (Neumann & Peters, 2008) sets Ψ as the return to go from state s_t. Critic Regularized Regression (CRR) (Wang *et al.*, 2020b) learns a Q value network from the dataset of collected experience and weighs samples in the BC objective by their Q value. Advantage Weighted Regression (AWR) (Peng, Kumar, Zhang & Levine, 2019) learns a network to estimate the advantage $A(s, a)$ of a state-action pair and sets $\Psi := e^{\beta \cdot A}$, where β is the temperature to control how much the advantage is weighted.

There has been recent work on improving generalization while remaining close to the dataset distribution so as to mitigate the effect of out of distribution (OOD) actions on learning. OOD actions can create instability in learning due to overestimation bias (Fujimoto, Meger & Precup, 2019). For example, when learning a policy to maximize the Q value, if the Q value is learnt from the data then there can be extrapolation error when querying the Q network with actions that were not seen in the data. Policy learning can exploit this vulnerability and learn to only produce such actions, leading to poor performance in the actual environment.

Conservative Q Learning (CQL) (Kumar, Zhou, Tucker & Levine, 2020) employs a regularization term on the Q network so that OOD actions are not overestimated. This is done through a penalty on the Q values of OOD actions. Implicit Q Learning (IQL) (Kostrikov, Nair & Levine, 2022) estimates an upper expectile of the Q network so that the maximum Q value of a state can be extracted without evaluating actions that are not present in the dataset distribution. Twin Delayed Deep Deterministic Policy Gradient + BC (TD3+BC) (Fujimoto & Gu, 2021) proposed a modification to an online off policy RL method, TD3 (Fujimoto *et al.*, 2018), to encourage policy behavior close to the dataset generating policy by adding a BC term to the policy optimization term of TD3 (hence the name TD3+BC). They found that this simple change was sufficient to utilize the online method in the offline setting. The policy optimization term and the BC objective are balanced using a trade off hyperparameter λ dependent on the scale of the Q values of a state. Batch Constrained Q Learning (BCQ) (Fujimoto *et al.*, 2019) learns a generative model that produces actions most similar to behavior seen in the offline dataset

and evaluates the Q value network only on these actions. The generative model introduces an implicit constraint to prevent querying the Q value network on OOD actions.

The Decision Transformer (DT) (Chen *et al.*, 2021) approaches offline RL from a different perspective. Instead of the objective being to learn a policy from the data, it is modeled as a sequence learning problem, termed upside-down RL (Schmidhuber, 2019). The key insight is to train a model that takes past states and actions along with the return to go from the current state and predicts the next action in the dataset. That is, you predict the next action conditioned on the return obtained from the current state. During evaluation, you condition the generation process on the maximum reward in the environment and the model would generate the sequence of actions to obtain that reward. This approach sidesteps the learning of a value function or a policy and instead frames offline RL solely as a conditional sequence modelling problem. They leverage a transformer (Vaswani *et al.*, 2017) for modelling sequences, given its recent success in sequence modelling problems in natural language processing.

CHAPTER 2

ORGANIZATION OF DOCUMENT

The primary portion of this thesis is the article "Bridging the Gap Between Offline and Online Reinforcement Learning Evaluation Methodologies" which has been submitted to the Transactions on Machine Learning Research journal in June 2023. This article introduces the sequential evaluation (SeqEval) framework for offline RL algorithms. It provides context on the current state of evaluation in offline RL, their deficiencies and how SeqEval addresses each of these concerns. We follow up with experimental results on a suite of benchmarks and algorithms, to show the ease with which SeqEval can be integrated in the current offline RL loop. Finally we present specific recommendations for how authors can summarize the results from SeqEval so that practitioners can quickly see the main takeaways.

We provide further discussion of the results in Chapter 4 and present the overall conclusions of the thesis in the Conclusions and Recommendations chapter.

CHAPTER 3

BRIDGING THE GAP BETWEEN OFFLINE AND ONLINE REINFORCEMENT LEARNING EVALUATION METHODOLOGIES

S. Sujit[1] , P. H. M. Braga[2] ,
, J. Bornschein[3] , S. E. Kahou[1]

[1] Département de génie logiciel et des technologies de l'information,
École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] Universidade Federal de Pernambuco,
Cidade Universitária, Recife - PE, 50670-901, Brazil
[3] DeepMind,
DeepMind, London, United Kingdom

Article submitted to Transactions on Machine Learning Research in July 2023

3.1 Introduction

Reinforcement learning (RL) has shown great progress in recent years with algorithms learning to play highly complex games with large state and action spaces such as DoTA2 ($\approx 10^4$ valid actions) and StarCraft purely from a reward signal of whether it won the game (Berner *et al.*, 2019; Vinyals *et al.*, 2019). However, each of these breakthroughs required a tremendous amount of environment interactions, sometimes upwards of 40 years of accumulated experience in the game (Schrittwieser *et al.*, 2019; Silver *et al.*, 2016, 2018). This can be infeasible for applications where such interactions are expensive, for example robotics.

Offline RL methods tackle this problem by leveraging previously collected data to bootstrap the learning process towards a good policy. These methods can obtain behaviors that maximize rewards obtained from the system conditioned on a fixed dataset of experience. The existence of logged data from industrial applications provides ample data to train agents safely till they achieve good performance and then can be trained on real hardware. The downside of relying on offline data without any interactions with the system is that the behavior learned can be limited by the quality of data available (Levine, Kumar, Tucker & Fu, 2020).

Levine *et al.* (2020) point out that there is a lack of consensus in the offline RL community on evaluation protocols for these methods. The most widely used approach is to train for a fixed number of epochs on the offline dataset and report performance through the average return obtained over a number of episodes in the environment. In this article, we propose to evaluate algorithms as a function of available data instead of just reporting final performance or plotting learning curves over a number of gradient steps. This approach allows us to study the sample efficiency and robustness of offline RL algorithms to distribution shifts while also making it easy to compare with online RL algorithms as well as intuitively study online fine-tuning performance. We call this approach of evaluation Sequential Evaluation (SeqEval) and present settings where SeqEval can be integrated into the evaluation protocol of offline RL algorithms. We also propose a style of model card (Mitchell *et al.*, 2018) that can be designed for offline RL algorithms to provide relevant context to practitioners about their sample efficiency to aid in algorithim selection.

3.2 Background and Related Work

3.2.1 Offline RL.

The simplest form of offline RL is behavior cloning (BC) which trains an agent to mimic the behavior present in the dataset, using the dataset actions as labels for supervised learning. However, offline datasets might have insufficient coverage of the states and operating conditions the agent will be exposed to. Hence BC agents tend to be fragile and can often perform poorly when deployed in the online environment. Therefore, offline RL algorithms have to balance two competing objectives, learning to generalize from the given dataset to novel conditions without deviating too far from the action distribution observed in the dataset in novel conditions. The former can be enabled by algorithms that are able to learn to identify and mimic portions of optimal behavior from episodes that are suboptimal overall, as pointed out in Chen *et al.* (2021). The latter issue is addressed by constraining the agent's policy around behavior seen in the dataset, for example, through enforcing divergence penalties on the policy distribution

(Peng *et al.*, 2019; Nair, Dalal, Gupta & Levine, 2020). Another way of penalizing out of dataset predictions is by using a regularizer on the Q value to prevent actions that have low support in the data distribution from having high Q values as done in Conservative Q Learning (CQL) (Kumar *et al.*, 2020). Decision Transformer (DT) (Chen *et al.*, 2021) does not learn a Q estimate and instead formulates the RL task as a sequence modelling task conditioned on the return to go from a given state. The agent is then queried for the maximum return and predicts actions to generate a sequence that produces the queried return. This is a very brief overview of offline RL methods, and we direct readers to Levine *et al.* (2020) for a broader overview of the field.

3.2.2 Metrics and Objectives.

The paradigm of *empirical risk minimization* (ERM) (Vapnik, 1991) is the prevailing training and evaluation protocol, both for supervised and unsupervised Deep Learning (DL). At its core, ERM assumes a fixed, stationary distribution and that we are given a set of (i.i.d.) data points for training and validation. Beyond ERM, and especially to accommodate non-stationary situations, different fields have converged to alternative evaluation metrics: In online learning, bandit research, and sequential decision making in general, the (cumulative) reward or the *regret* are of central interest. The regret is the cumulative loss accrued by an agent relative to an optimal agent when sequentially making decisions. For learning in situations where a single, potentially non-stationary sequence of observations is given, Minimum Description Length (MDL)(Rissanen, 1984) provides a theoretically sound approach to model evaluation. Multiple, subtly different formulations of the description length are in use, however, they are all closely related and asymptotically equivalent to *prequential MDL*, which is the cumulative log loss when sequentially predicting the next observation given all previous ones (Rissanen, 1987; Poland & Hutter, 2005). Similar approaches have been studied and are called the *prequential approach* (Dawid & Vovk, 1999) or simply *forward validation*. A common theme behind these metrics is that they consider the agents' ability to perform well, not only in the big-data regime, but also its generalization performance at the beginning, when only a few observations are available for learning. The MDL literature provides arguments and proofs why those models,

Algorithm 1: Outline of SeqEval

Input: Algorithm A, Offline data $\mathcal{D} = \{s_t, a_t, r_t, s_{t+1}\}_{t=1}^{T}$, increment-size γ, gradient steps per increment K
Replay-buffer $\mathcal{B} \leftarrow \{s_t, a_t, r_t, s_{t+1}\}_{t=1}^{T_0}$
$t \leftarrow T_0$
while $t < T$ **do**
    Update replay-buffer $\mathcal{B} \leftarrow \mathcal{B} \cup < s_t, a_t, r_t, s_{t+1} >_{t}^{t+\gamma}$
    Sample a training batch, ensure new data is included: batch $\sim \mathcal{B}$
    Perform training step with A on batch.
    $t \leftarrow t + \gamma$
    for $j = 1, \cdots, K$ **do**
        Sample a training batch $\sim \mathcal{B}$
        Perform training step with A on batch.

that perform well in the small-data regime without sacrificing their big-data performance, are expected to generalize better to future data (Rathmanner & Hutter, 2011).

With these two aspects in mind, that a) RL deals with inherently non-stationary data, and b) that sample efficiency is a theoretically and practically desirable property, we propose to evaluate offline RL approaches by their data efficiency.

3.3 Sequential Evaluation of Offline RL Algorithms

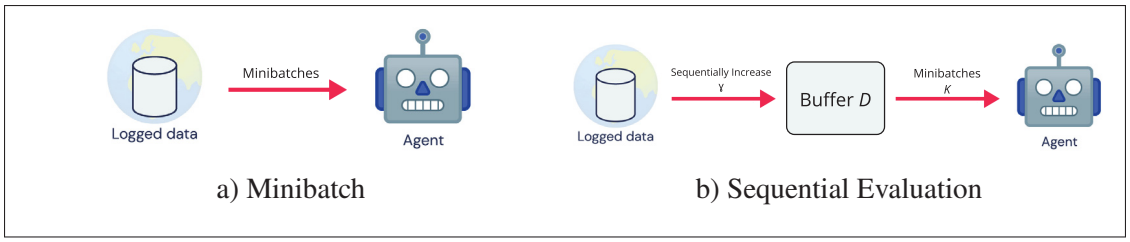


Figure 3.1 Comparison of traditional training schemes with Sequential Evaluation

As mentioned above, one approach for offline RL evaluation is to perform multiple epochs of training over the dataset. We contend that there are a few issues with this approach. Firstly,

this approach does not provide much information about the sample efficiency of the algorithm since it is trained on all data at every epoch. This means that practitioners do not see how the algorithm can scale with the dataset size, or if it can achieve good performance even with small amounts of logged data. Furthermore, there can be distribution changes in the quality of the policy in the dataset, and evaluating as a function of epochs hides how algorithms react to these changes. Finally, there is a disconnection in the evaluation strategies of online and offline RL algorithms, which can make it difficult to compare algorithms realistically.

Instead of treating the dataset as a fixed entity, we propose that the portion of the dataset available to the agent change over time and that the agents' performance is evaluated as a function of the available data. We term this as sequential evaluation of offline algorithms (SeqEval). This can be implemented by reusing any of the prevalent replay-buffer-based training schemes from online deep RL. But instead of extending the replay-buffer with sampled trajectories from the currently learned policy, we instead slowly insert prerecorded offline RL data. We alternate between adding new samples to the buffer and performing gradient updates using mini-batches sampled from the buffer. The number of samples added to the buffer at a time is denoted by γ and the number of gradient steps performed between each addition to the buffer is denoted by K. A concrete implementation of the approach is outlined in Alg. 1.

This approach of evaluation addresses several of the issues with epoch-style training. By varying γ and K we can get information about the scaling performance of an algorithm with respect to dataset size, which tells us if data is the bottleneck for further improvements, and how quickly the algorithm can learn with limited data. We can visualize how the algorithm behaves with shifts in dataset quality directly from the performance curves. There is a direct analogy for evaluation in the online RL setting since online methods are evaluated as a function of the number of environment steps, which is a measure of the amount of data it has access to in the replay buffer and hence can be directly connected to the size of the replay buffer in the offline method. We can also seamlessly evaluate the performance of the algorithm in online fine-tuning by adding samples from the environment once the entire offline dataset is added to the replay buffer. A benefit of the sequential approach is that it does not require a complete overhaul in

codebases that follow existing training paradigms. For the baselines that we present in this article, we were able to use the sequential evaluation approach with less than 10 lines of changes to the original codebases.

The ratio $\frac{K}{\gamma}$ is the replay ratio (RR), a hyperparameter commonly found in RL algorithms which use replay-buffers. The RR determines how many gradient steps are performed for each added data point. Increasing RR allows some algorithms to extract more utility from the data it has received, potentially improving sample efficiency. By varying RR we can identify whether computation is the bottleneck for improving performance, or the amount of available data.

3.3.1 Implementation Details

To ensure that the algorithm sees each data point in the dataset at least once, when a new batch of data is added to the buffer, the algorithm is trained on that sample of data once before K mini-batches are sampled from the buffer for training. If the batch added is smaller than the mini-batch size, then it can be made part of the next mini-batch that is trained on. In practice, we found that setting γ and K to 1 worked well in all datasets tested. This means that the x-axis of all plots directly corresponds to the number of samples available for training and the number of gradient updates performed. Additionally, we study other values of the RR and report these results in Appendix 2. We observe that a RR of 1 generally performs well across datasets and higher RRs lead to overfitting and worse performance. The changes made to the codebase of each algorithm are as follows: Each codebase had a notion of a replay buffer that was being sampled, and the only addition required here was a counter that kept track of up to which index in the buffer data points could be sampled from to create mini-batches. The counter was initialized to $T_0 = 5000$ so that there were some samples in the buffer at the start of training. The second change that needed to be made was changing the outer loop of training from epochs to the number of gradient updates and incrementing the buffer counter by γ every K update. This way, the amount of data the algorithm was trained on sequentially increased to the full dataset over the course of training. Finally, we shuffle the order of the dataset on each seed so that the training

curves are not specific to only one particular ordering of the dataset. From our experiments we see that the dataset order does not have a significant impact on the performance of the algorithm.

3.4 Experiments

3.4.1 Baselines and Benchmarks

We evaluate several existing offline RL algorithms using the sequential approach, namely IQL (Kostrikov *et al.*, 2022), CQL (Kumar *et al.*, 2020), TD3+BC (Fujimoto & Gu, 2021), AWAC (Nair *et al.*, 2020), BCQ (Fujimoto *et al.*, 2019), Decision Transformer (DT) (Chen *et al.*, 2021) and Behavior Cloning (BC). These algorithms were evaluated on the D4RL benchmark (Fu, Kumar, Nachum, Tucker & Levine, 2020), which consists of three environments: Halfcheetah-v2, Walker2d-v2 and Hopper-v2. For each environment, we evaluate four versions of the offline dataset: random, medium, medium-expert, and medium-replay. Random consists of 1M data points collected using a random policy. Medium contains 1M data points from a policy that was trained for one-third of the time needed for an expert policy, while medium-replay is the replay buffer that was used to train the policy. Medium-expert consists of a mix of 1M samples from the medium policy and 1M samples from the expert policy. These versions of the dataset are useful for evaluating the performance of offline agents across a wide spectrum of dataset quality.

We also created a dataset from the DeepMind Control Suite (DMC) (Tassa *et al.*, 2018) environments following the same procedure as outlined by the authors of D4RL. We chose the DMC environments because of their high dimensional state space and action space. Specifically we chose cheetah run, walker run and finger turn hard and trained a Soft Actor Critic (SAC) agent (Haarnoja, Zhou, Abbeel & Levine, 2018) on these environments for 1M steps. SAC is chosen because it is able to solve these tasks in 1M environment steps. A medium and expert version of the dataset was created for each of the three environments using policies obtained after 500K and 1M steps respectively.

3.4.2 Experimental Settings

We study varied ways in which SeqEval can be integrated into offline RL training. The first setting is the Standard setting in which samples are added periodically to the buffer during training. The results of the Standard setting on a subset of datasets are given in Fig. 3.2 and the complete set of datasets, along with an experiment comparing curves with larger K are available in the appendix. Secondly, to highlight how SeqEval can visualize how the algorithm reacts to changing dataset quality during training, we create a "Distribution Shifts" setting where a mixed version of each environment is created. In this dataset, the first 33% of data comes from the random dataset, the next 33% from the medium dataset and the final 33% from the expert dataset. Each algorithm is sequentially given samples and from the performance curves given in Fig. 3.5 we can see how each algorithm adapts to changes in the dataset distribution. We then show how SeqEval can be utilised in multi task offline RL in a similar fashion. Instead of providing the entire dataset of a new task to the agent at once, we periodically add new samples of the task to the agent over time. This lets us study the amount of samples required for the agent to transfer to the new task. Finally we also show how SeqEval supports seamless integration of online fine-tuning experiments into performance curves. In this setting, once the entire offline dataset is added to the replay buffer, the agent is allowed to interact with the online simulator for a fixed number of steps (500k steps in our experiments). Since the curves are a function of data samples, we can continue evaluating performance as before. The results on a subset of datasets are given in Fig. 3.7, and curves for all datasets are available in the appendix.

For each dataset, we train algorithms following Alg. 1, initializing the replay buffer with 5000 data points at the start of training. We set γ and K each to 1, that is, there is one gradient update performed each time a sample is added to the buffer. The results are presented in Fig. 3.2, where the x-axis represents the amount of data in the replay buffer. In each plot, we also include the performance of the policy that generated the dataset as a baseline, which provides context for how much information each algorithm was able to extract from the dataset. This baseline is given as a horizontal dotted line.

3.4.3 Model Cards

Since SeqEval evaluates algorithms as a function of data, it opens up the possibility of quantitatively analysing how algorithms scale with data. We propose a few additional evaluation criteria in addition to the training curves. The first one is Perf@50%, which measures the aggregate performance of algorithms when half the dataset is available to it for training. To further analyse the role of data scaling, we studied two additional measures, Perf@50% normalised with respect to performance obtained with the whole dataset (Perf@100%) and the difference between Perf@100% and Perf@50%. These three metrics give end users valuable insights that can aid in offline algorithm selection. They can be presented as part of an overall model card (Mitchell *et al.*, 2018) available to the user.

3.4.4 Results

3.4.4.1 Standard

We present the results of the Standard setting in Figs. 3.2 and 3.3. One striking observation from Fig. 3.2 is how quickly the algorithms converge to a given performance level and then stagnate. This is most evident in the medium version of each environment. With less then 300K data points in the buffer, each algorithm stagnates and does not improve in performance beyond that point even after another 500K points are added. There are diminishing returns from adding data beyond 500K points to the buffer. This highlights that most of the tested algorithms are not very data-hungry. That is, they do not require a large data store to reach good performance, which is beneficial when they need to be employed in practical applications. The experiment highlights that the chosen datasets might lack diversity in collected experience since most algorithms appear to need only a fraction of it to attain good performance. This phenomenon is made more stark in Fig. 3.4. Most algorithms reach within 90% of final performance with just 50% of the data. In other words, doubling the data in the D4RL benchmarks provides an aggregated uplift of 10% at best. This indicates that more effort could be put into designing a harder benchmark.

The effect is less pronounced in the DMC datasets, where the algorithms continue learning till about 50% of the dataset is added and then learning begins to stagnate.

In a majority of the datasets tested on D4RL, CQL reaches better performance than other methods earlier in training and consistently stays at that level as more data is added. TD3+BC exhibits an initial steep rise in performance but levels out at a lower score overall, or in the case of Halfcheetah-medium-expert, degrades in performance as training progresses.

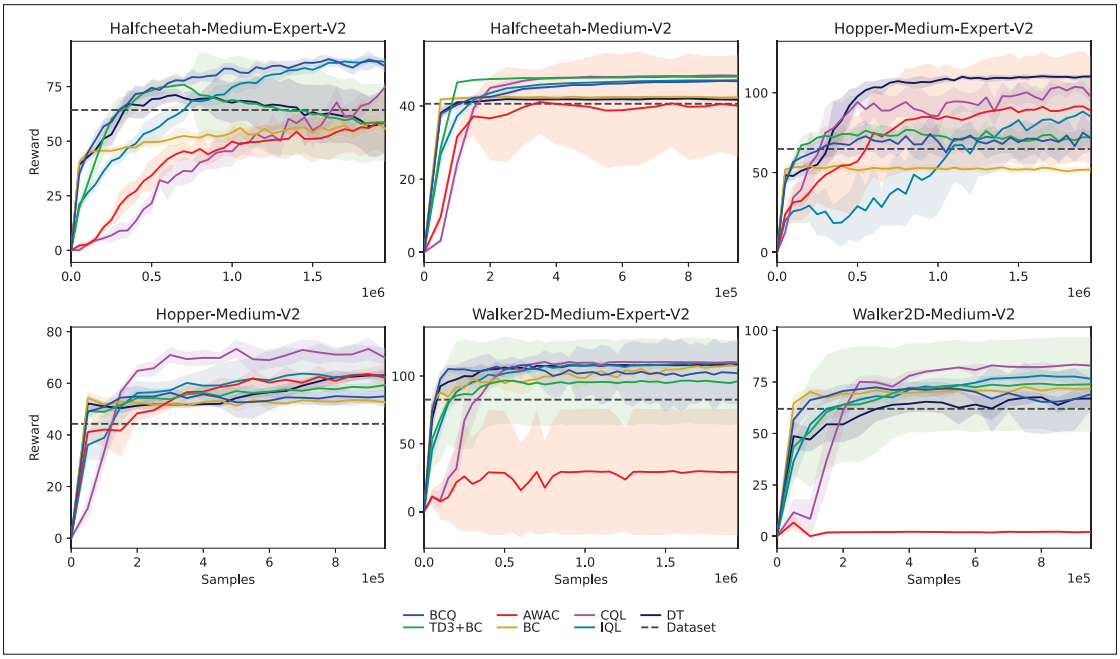


Figure 3.2 Performance curves on the D4RL benchmark as a function of data points seen. Shaded regions represent standard deviation across 5 seeds.

3.4.4.2 Distribution Shifts

The learning curves for the "mixed" dataset in the D4RL benchmark and DMC are given in Figs. 3.5 and 3.6. In the mixed dataset experiment, while most of the algorithms do adapt to the new data, continually improving each time there is a shift in the dataset quality, though rarely are they able to do so in every single environment. While CQL adapts quickly in both HalfCheetah and Walker2d, it fails to learn at all in Hopper. TD3+BC outperforms all other

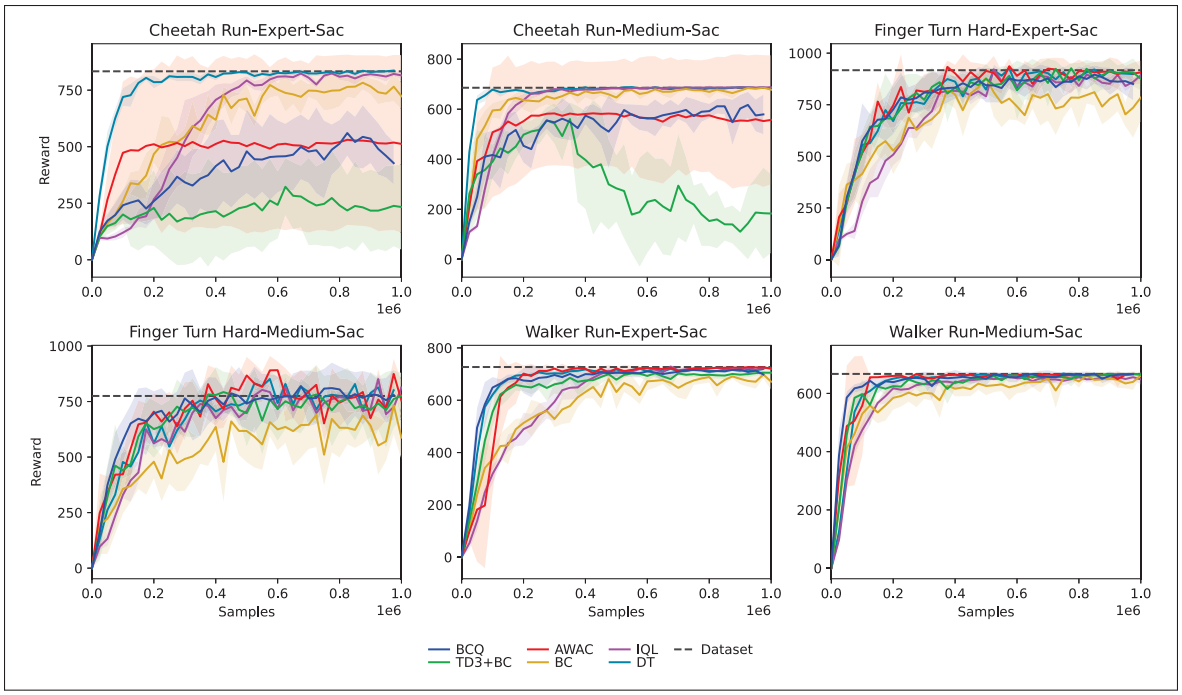


Figure 3.3 Performance curves on the DMC Dataset as a function of data points seen. Shaded regions represent standard deviation across 5 seeds.

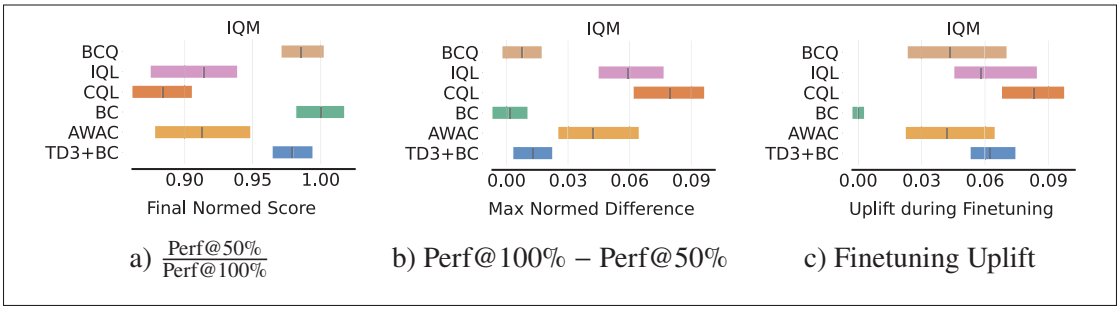


Figure 3.4 a) and b): Analysis of data scaling in the D4RL benchmark. Perf@X% refers to performance when the algorithm has seen X% of the dataset. c) Comparison of the increase in performance each algorithm achieves after 500k simulator steps. Aggregated across 12 datasets and 3 seeds.

methods in HalfCheetah, but is the worse performing in Walker2d. DT is able to consistently adapt to new data in each environment. A surprising result is how well BC performs in each environment, with BC nearly being the second best performing algorithm in all environments.

We hypothesise that since there is good coverage of the state spaces and optimal behavior in the dataset, behavior cloning is able to learn effectively. It does not have to "stitch" behavior from different episodes and instead just need to replicate the behavior in the dataset. This is in line with the findings of (Chen *et al.*, 2021) which found that 10% BC, a BC variant trained on only the top 10% percentile of episodes (based on episodic return), was competitive with specialised offline algorithms on the D4RL benchmark.

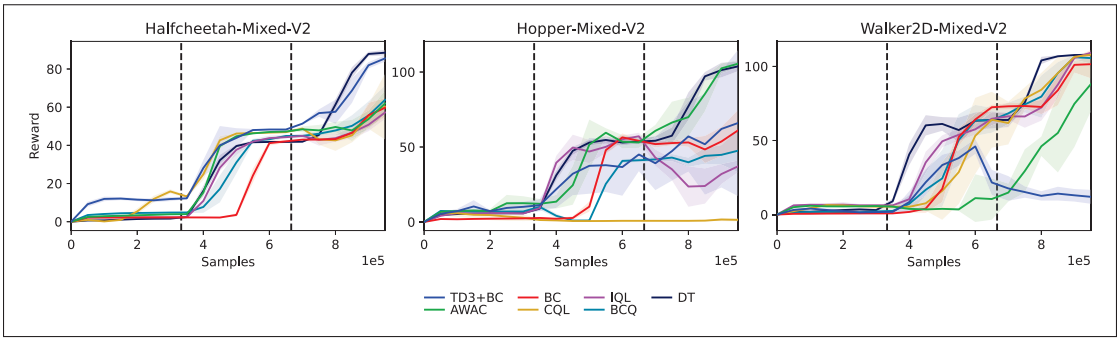


Figure 3.5 Performance curves for Mixed version of D4RL with varying dataset quality. Dotted line indicates where there is a change in the dataset generating policy distribution. Horizontal dotted line indicates the performance of the policy that generated the data. Shaded regions represent standard deviation across 5 seeds.

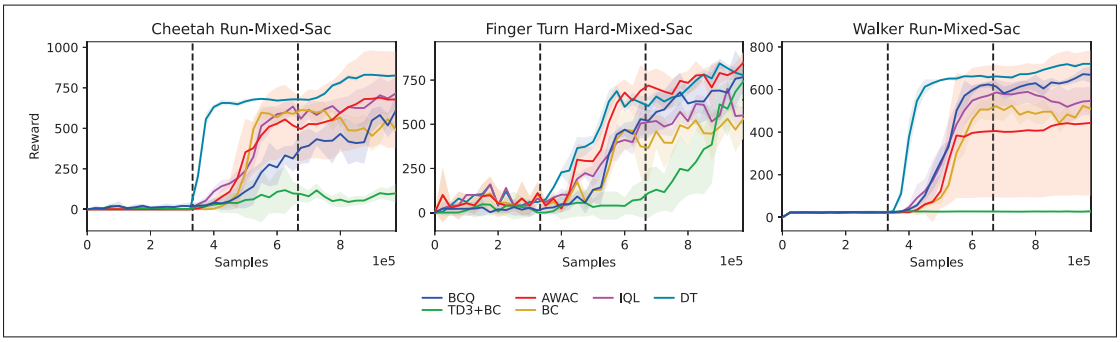


Figure 3.6 Performance curves for Mixed version of DMC with varying dataset quality. Dotted line indicates where there is a change in the dataset generating policy distribution. Horizontal dotted line indicates the performance of the policy that generated the data. Shaded regions represent standard deviation across 5 seeds.

3.4.4.3 Online Finetuning

In online fine-tuning, shown in Fig. 3.7 and Table I-2, CQL reaches higher scores compared to the other algorithms showing the versatility of CQL in handling both offline datasets and online interactions with the system. From Fig. 3.4c, we see that in general there are modest gains from online finetuning in these algorithms.

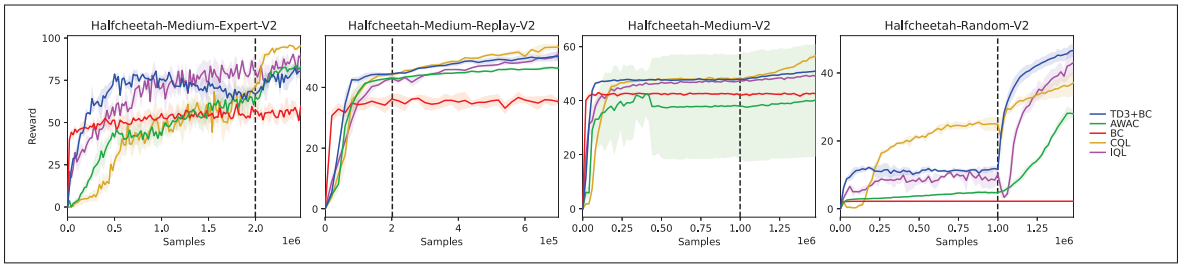


Figure 3.7 Performance curves for online fine-tuning. Each algorithm is given 500k steps in the simulator after sequential evaluation of the offline dataset. Dotted line indicates where online fine-tuning begins. Shaded regions represent standard deviation across 3 seeds.

3.4.5 Comparison with Mini Batch Style Training

We compare how algorithms perform when given access to the same amount of compute in the SeqEval setting and the mini batch setting. The results are given in Fig. 3.8. In the mini batch style training, the algorithm is given access to the entire dataset at all times during training. The x axis in Fig. 3.8 is the number of gradient steps, while in the previous figures for SeqEval the x axis represents the number of data points available for training. However when $\gamma = 1$ and $K = 1$, the x axis can also represent the number of gradient steps, hence we plot the SeqEval and mini batch training curves together. So while the algorithm has performed a similar amount of computation when comparing the curves at a given gradient step, the data they have access to at these points can be very different. As a result, SeqEval would exhibit a slower rise in performance per gradient step compared to mini batch, but that is because it simply does not

have access to the same amount of data as mini batch style training. We do see that SeqEval closes the gap towards the end of training.

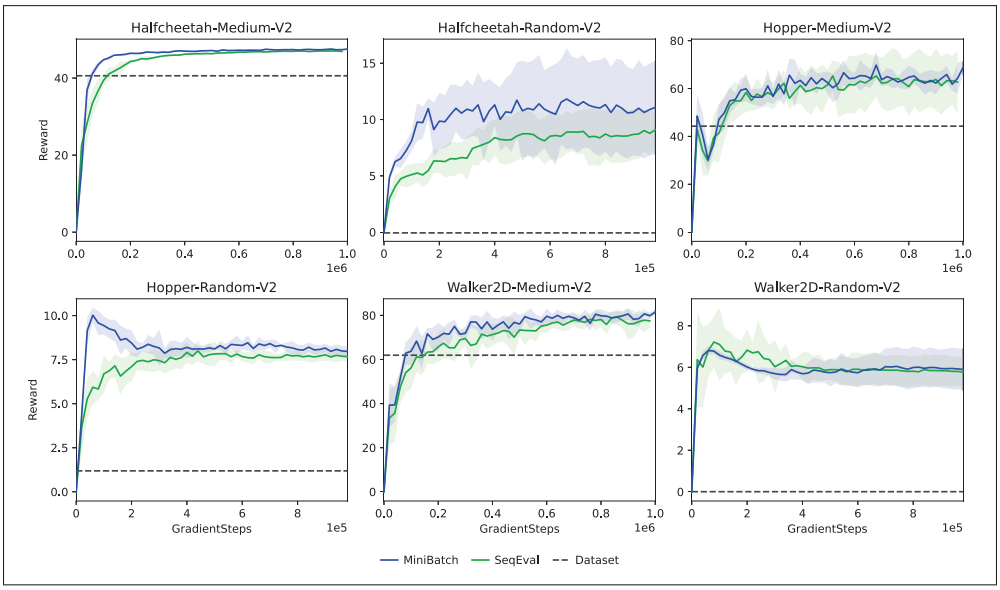


Figure 3.8 Comparison of SeqEval training and mini batch style training.

3.4.6 Additional Metrics for Evaluation

While we have used the evaluation episodic return as the performance metric in our experiments, it is not necessary to do so. For example, if there are environments where a simulator is not available for periodic cheap evaluation, off-policy evaluation metrics (Thomas, Theocharous & Ghavamzadeh, 2015; Wang, Gao & Zha, 2020a; Munos & Szepesvári, 2008) can be used. We provide an example of this in Fig. 3.9 where instead of monitoring the episodic return during training, we utilize Fitted Q Evaluation (FQE) score (Wang *et al.*, 2020a). We followed the same implementation as given in Wang *et al.* (2020a), training the FQE estimator from scratch each time during an evaluation phase. The FQE estimator was given access to the entire dataset during its training, while IQL still followed the SeqEval style of training. The exact FQE metric used was $\mathbb{E}[\hat{Q}(s_0, \pi(s_0))]$, i.e. the predicted Q value of the start state distribution.

We would like to emphasize that SeqEval is independent of the evaluation metric used during training. SeqEval is a style of training and can be used with different evaluation metrics depending on the constraints, for example simulator return or off-policy evaluation metrics such as FQE scores. The experiment above is a concrete example of how SeqEval can be used in situations where there is limited access to a simulator to evaluate performance.

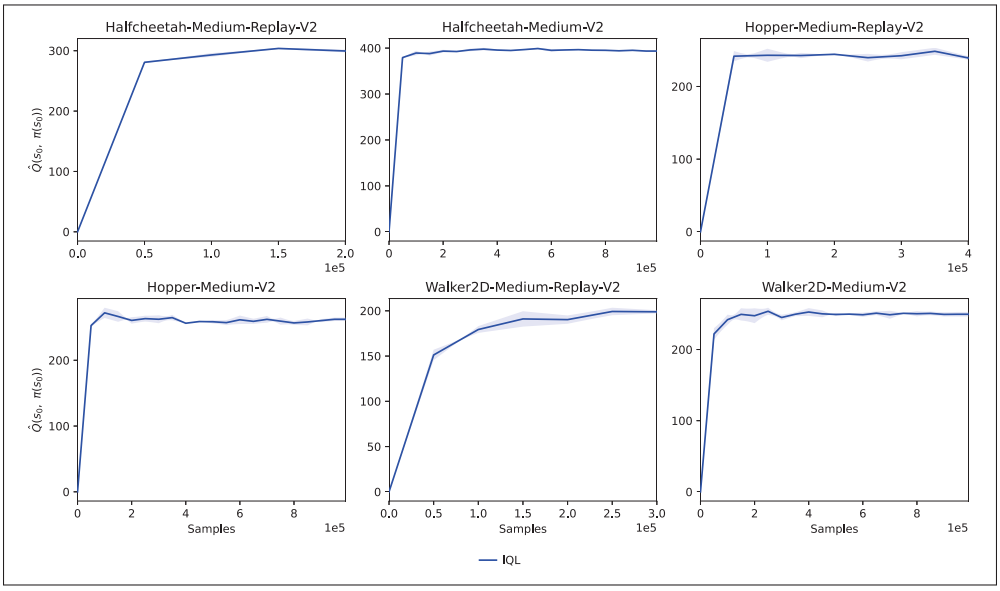


Figure 3.9 Performance of IQL in the SeqEval setting using FQE as the evaluation metric.

3.5 Conclusion

In this paper, we propose a sequential style of evaluation for offline RL methods so that algorithms are evaluated as a function of data rather than compute or gradient steps. In this style of evaluation, data is added sequentially to a replay buffer over time, and mini-batches are sampled from this buffer for training. This is analogous to online training to deep RL and allows us to measure the data scaling and robustness of offline algorithms simultaneously from the training curves. We compared several existing offline methods using sequential evaluation and showed how their training curves allow for algorithm selection depending on data efficiency or performance. Finally we compared sequential evaluation to traditional mini batch style training to highlight the difference in training dynamics induced by each scheme. We believe

that sequential evaluation holds promise as an established method of evaluation for the offline RL community.

CHAPTER 4

DISCUSSION OF RESULTS

In this chapter we review the main results of the thesis. We created an instantiation of SeqEval in two benchmark suites and compared 7 different algorithms across three experimental settings. The first setting studied was the standard setting of SeqEval, where we used $\gamma = 1$ and $K = 1$, that is for every data point that was added to the buffer, one mini-batch was sampled and trained on. From the learning curves in Figs. 3.2 and 3.3 we see that even though the dataset contains 1M data points, most algorithms converge in performance within 500K data points. Among the methods tested, CQL consistently was the best performing algorithm. We were surprised that BC performed as well as it did and believe this is due to the good state coverage in the dataset, allowing the agent to be robust. The second setting was the distribution shifts setting which tested algorithms on their ability to continuously adapt to new data of higher quality. DT was by far the best performing algorithm across the 6 environments tested, highlighting the strength of the transformer architecture. Finally we looked at the online finetuning setting, which has also been proposed for mini-batch style training Kostrikov *et al.* (2022); Nair *et al.* (2020). TD3+BC and CQL were both competitive in these experiments. Even when pretrained on random data, algorithms are able to quickly improve once given access to the actual environment.

CONCLUSION AND RECOMMENDATIONS

In this thesis, we presented SeqEval, a framework for training and evaluating offline RL algorithms with an emphasis on studying their data scaling properties. Our research project started by identifying drawbacks in the current standard of minibatch style training in offline RL, followed by formulating the research problem to devise better evaluation strategies for offline RL to mitigate these drawbacks. To this aim, we proposed that instead of the agent being given access to the entire dataset at all times during training, we incrementally increase this training buffer. That way, over the course of training, the agent is exposed to more and more data, allowing us to visualize the performance curves with respect to dataset size rather than gradient steps.

A surprising insight from our experiments was how the algorithms studied learnt with a fraction of the dataset size and learning stagnated after 20-30% of the dataset was added. Further addition of data did not lead to improvements in performance. This observation was consistent across environments and algorithms. We can view these results in two ways.

One, the current benchmarks/datasets for offline RL algorithms need to be re-evaluated. If an algorithm can perform similarly with 20% of the data or 100% of the data, then these datasets might not be sufficiently difficult to properly evaluate algorithms. We believe that effort should be invested in designing harder benchmarks for this domain.

The second takeaway is that the algorithms tested were surprisingly data efficient. The model cards in Figs. 3.4a and 3.4b show that most algorithms reach within 90% of final performance with less than 50% of the dataset. Presenting such summary statistics can make it easy for practitioners to 1) perform algorithm selection, and 2) evaluate the benefits of additional data collection. The latter can be useful in practical domains where data collection is expensive and time consuming and the model cards can provide insights into the tradeoff between better algorithmic performance and increased cost of data collection.

We encourage the offline RL community to utilize SeqEval style training when designing novel algorithms and believe that the performance curves and model cards can be effective ways of highlighting the strength of their algorithms. As mentioned previously, future work could look at alternative evaluation metrics instead of the episodic return for performance curves so that it can be applied in domains where it is difficult to execute the agent in the environment itself. This could be through off policy evaluation metrics such as FQE scores (Thomas *et al.*, 2015).

APPENDIX I

ADDITIONAL RESULTS

1. Performance on D4RL Benchmark

We present complete training curves on all twelve datasets that were used in Fig. I-2 and final performance in Table I-1. In addition to the curves, we compare the algorithms at the end of training with scores aggregated across environments. This is done using the rliable (Agarwal, Schwarzer, Castro, Courville & Bellemare, 2021) library to plot interval estimates of normalized performance measures such as median, mean, interquartile mean (IQM) and optimality gap. The optimality gap is a measure of how far an algorithm is from optimal performance aggregated across environments. So, lower values are better. The scores in each dataset are normalized with respect to the maximum score, which is 100. These results are given in Fig. I-4. In both the performance curves and aggregated scores we can see that CQL outperforms other tested methods by a clear margin. A curious phenomenon observed was that AWAC (Nair *et al.*, 2020) is unable to learn at all in the Walker2d environment with either the medium-expert or the medium version achieving very low rewards. As a sanity check we set γ to the size of the dataset and reran experiments and the method achieved results similar to those reported in the paper, indicating it was not an implementation problem. This is surprising since AWAC is proposed as an algorithm that can work for online fine-tuning following offline pretraining, but among all the methods tested, it had drastic changes in performance when using sequential evaluation. The final performance in the online fine-tuning task and the mixed version of the environment is also given in Tables I-2 and I-3. The Perf@50% and Perf@100% results are given in Fig. I-1

Moreover, Fig. I-5 show how the algorithms performed when K was set to 2. This experiment studied if we had not trained the methods for enough gradient steps and if additional performance could be extracted from the data. However, performance remained essentially the same or even degraded in some instances, showing that this was not the case.

2. Effect of Replay Ratio in SeqEval

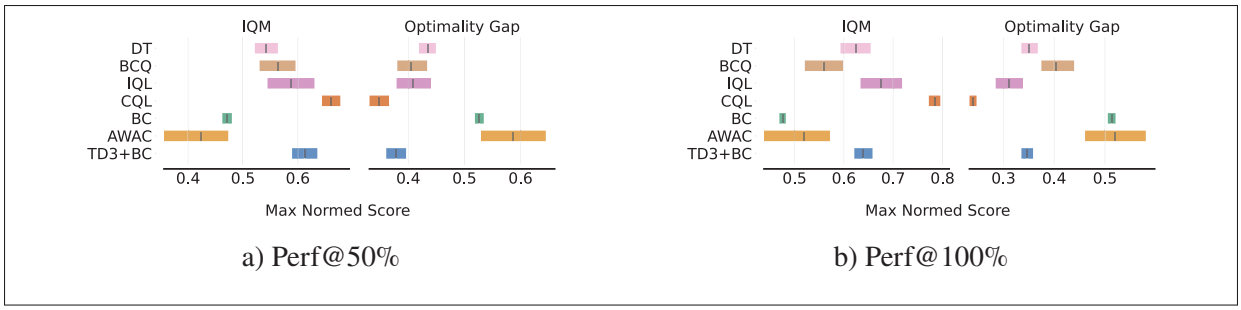


Figure-A I-1 Comparison of performance when a) half the dataset is available b) full dataset is available.

Table-A I-1 Performance of each algorithm on the D4RL Benchmark

Dataset	BCQ	TD3+BC	AWAC	BC	CQL	DT	IQL
halfcheetah-medium-expert-v2	92.25	6.83	59.04	63.44	83.45	71.6	86.0
halfcheetah-medium-replay-v2	37.54	44.26	43.37	36.3	43.54	30.91	41.97
halfcheetah-medium-v2	45.47	48.45	46.58	43.12	49.13	42.65	46.96
halfcheetah-random-v2	8.16	10.09	2.26	2.26	24.43	2.12	8.41
hopper-medium-expert-v2	108.8	69.92	111.88	44.53	111.0	111.38	49.18
hopper-medium-replay-v2	23.9	63.04	65.47	14.02	88.51	74.94	69.09
hopper-medium-v2	53.39	49.77	52.8	55.84	70.53	62.06	67.49
hopper-random-v2	7.16	8.13	9.18	2.26	6.2	7.41	7.63
walker2d-medium-expert-v2	110.36	108.33	1.98	107.58	109.98	108.23	98.51
walker2d-medium-replay-v2	59.94	76.8	82.54	24.63	73.31	55.04	63.12
walker2d-medium-v2	76.85	83.4	1.76	78.78	83.16	65.79	80.85
walker2d-random-v2	0.11	0.49	3.48	0.63	-0.12	2.48	7.82

The experiments in the paper studied a replay ratio (RR) of 1. The results are in Figs. I-6 and I-7. We can directly compare the training and evaluation curves for different RRs and conclude whether compute (gradient-steps) or training data are the limiting factor. In our main experiments we study the setting where RR = 1. We utilized this setting since we observed that increasing the compute budget did not improve performance. Lower RRs do not extract as much information as possible from the dataset and do worse. These curves provide valuable insights into the combined data and compute scaling of an algorithm.

Table-A I-2 Performance of each algorithm in the online fine-tuning task on the D4RL Benchmark

Dataset	BCQ	TD3+BC	AWAC	BC	CQL	IQL
finetune-halfcheetah-medium-expert-v2	88.02	74.36	83.6	61.77	96.72	87.74
finetune-halfcheetah-medium-v2	47.86	51.9	52.89	42.86	55.29	49.24
finetune-halfcheetah-random-v2	22.29	48.53	30.67	2.26	34.17	50.87
finetune-hopper-medium-expert-v2	43.19	112.71	111.9	48.01	100.25	110.66
finetune-hopper-medium-v2	52.35	63.03	45.45	58.92	94.31	73.59
finetune-hopper-random-v2	14.31	9.85	8.78	2.47	4.56	12.05
finetune-walker2d-medium-expert-v2	108.28	107.77	1.41	108.35	110.73	110.81
finetune-walker2d-medium-v2	71.26	85.01	21.32	66.47	83.7	84.24
finetune-walker2d-random-v2	1.42	8.6	1.67	0.5	0.33	10.56

Table-A I-3 Performance of each algorithm in the mixed version of the D4RL Benchmark

Dataset	BCQ	TD3+BC	AWAC	BC	CQL	DT	IQL
halfcheetah-mixed-v2	66.98	80.65	29.55	57.38	93.47	81.51	68.04
hopper-mixed-v2	55.29	112.42	110.36	86.67	0.75	111.74	51.37
walker2d-mixed-v2	102.77	8.15	98.36	108.79	109.71	107.63	109.83

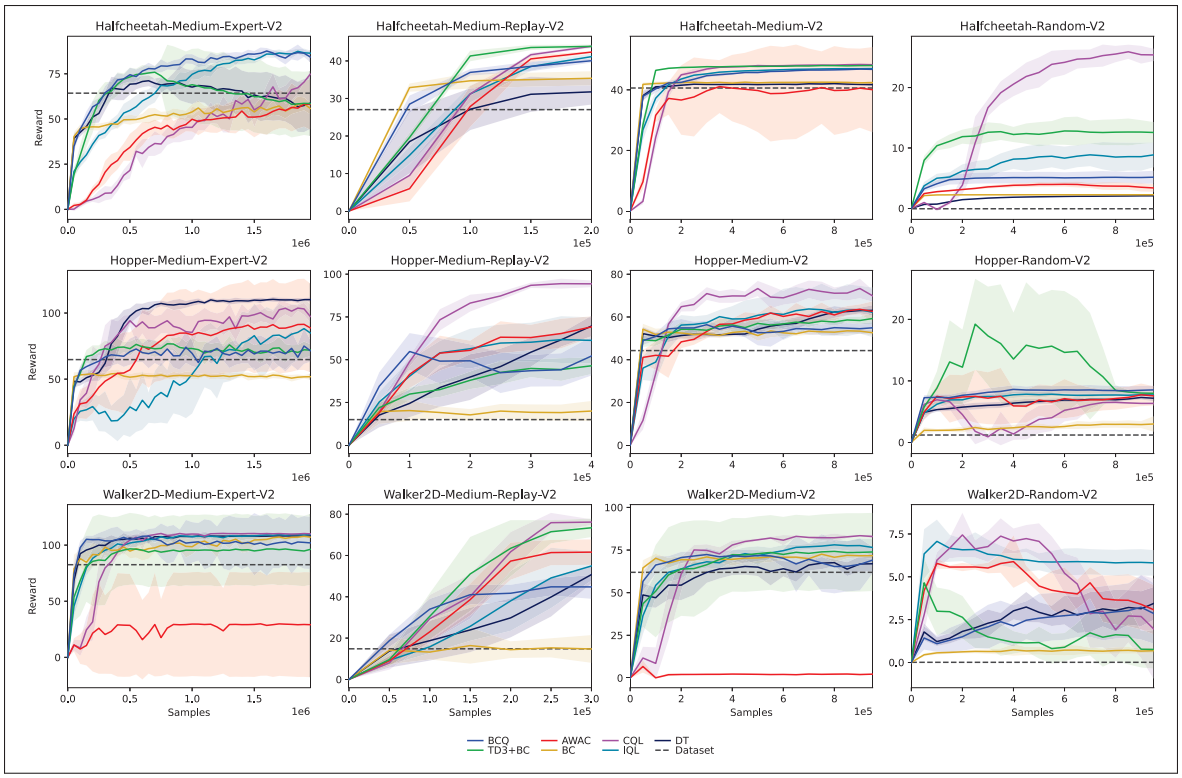


Figure-A I-2 Performance curves on the D4RL benchmark of offline RL algorithms as a function of data points seen. Shaded regions represent standard deviation across 5 seeds.

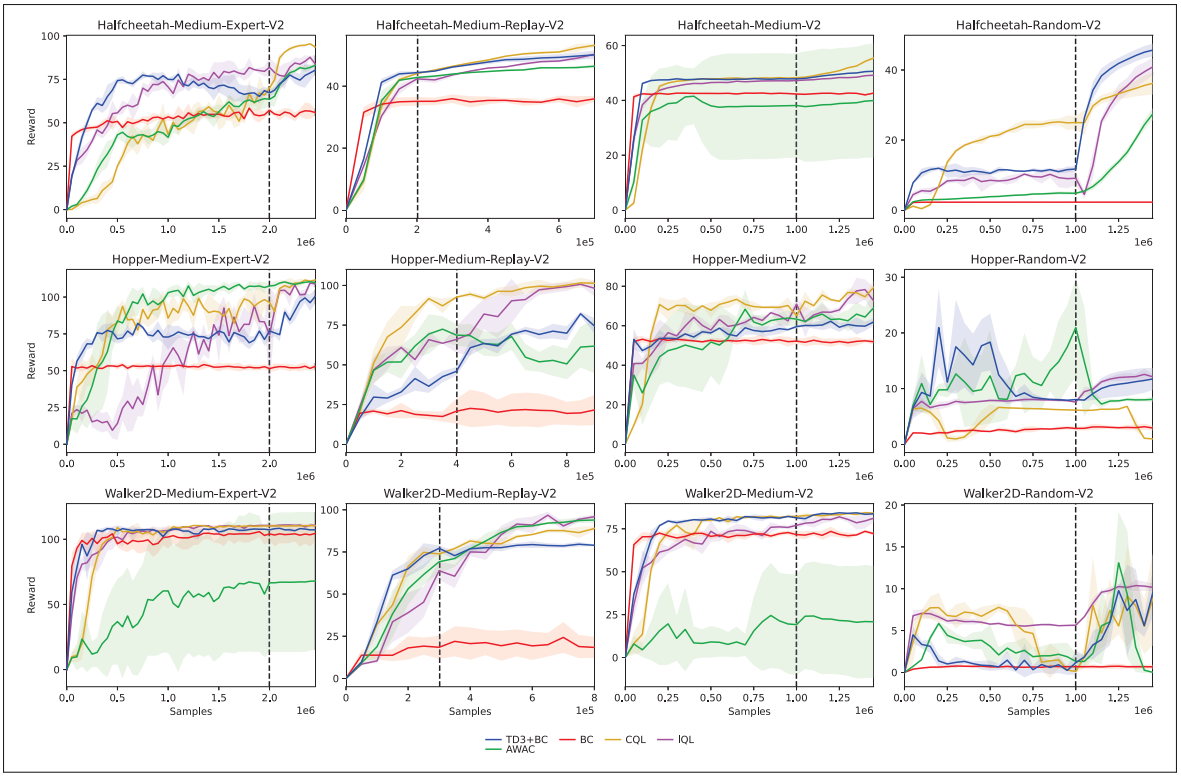
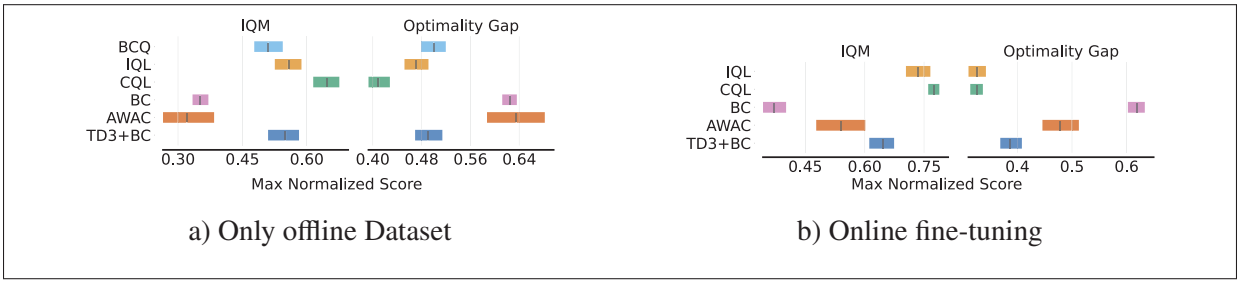


Figure-A I-3 Performance curves for online fine-tuning. Each algorithm is given 500k steps in the simulator after sequential evaluation of the offline dataset. Dotted line indicates where online fine-tuning begins. Shaded regions represent standard deviation across 3 seeds.



a) Only offline Dataset

b) Online fine-tuning

Figure-A I-4 Performance aggregated across environments using rliable. For IQM higher is better, while for Optimality gap, lower is better.

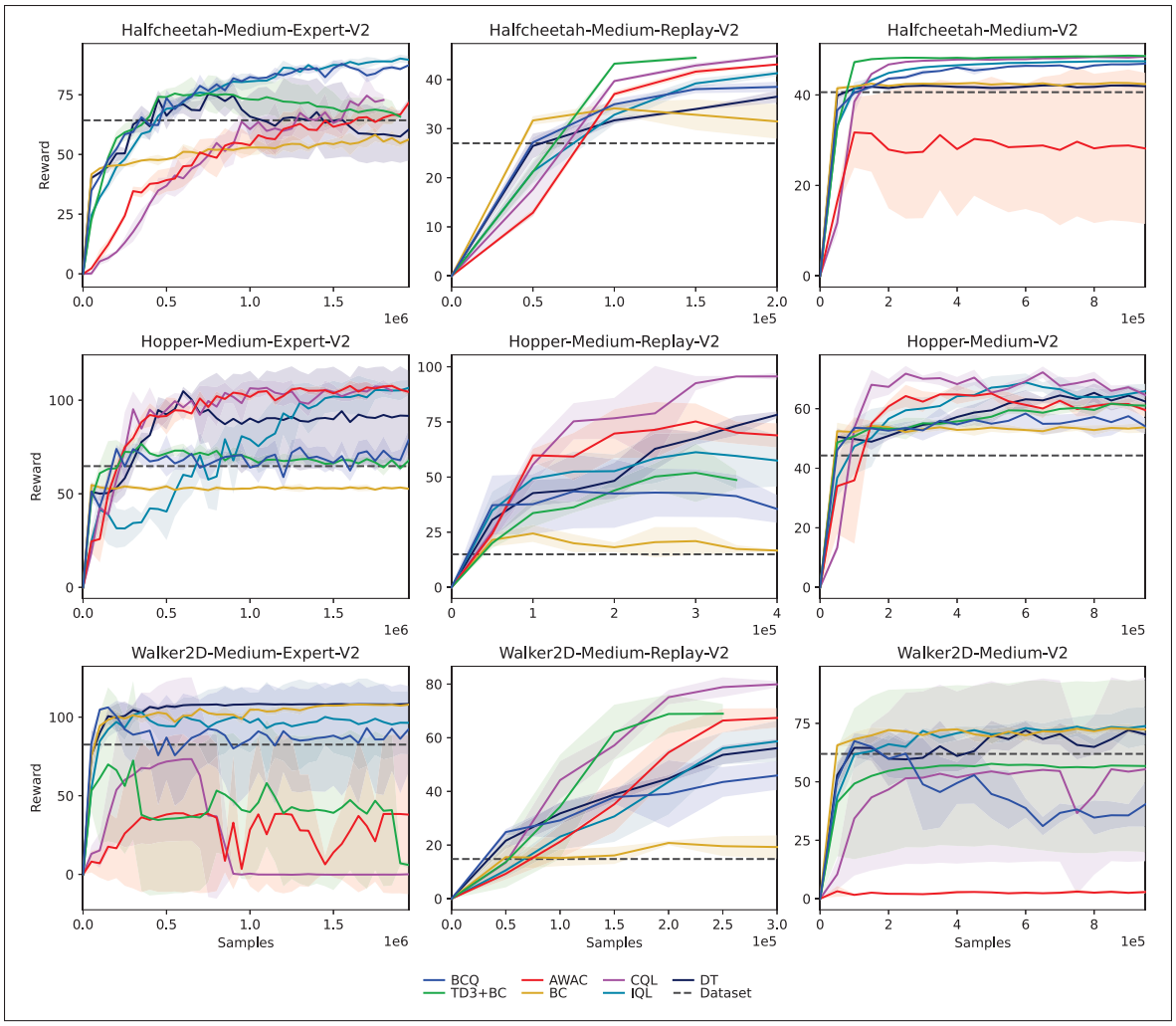


Figure-A I-5 Performance curves on the D4RL benchmark with K increased to 2. Shaded regions represent standard deviation across 3 seeds.

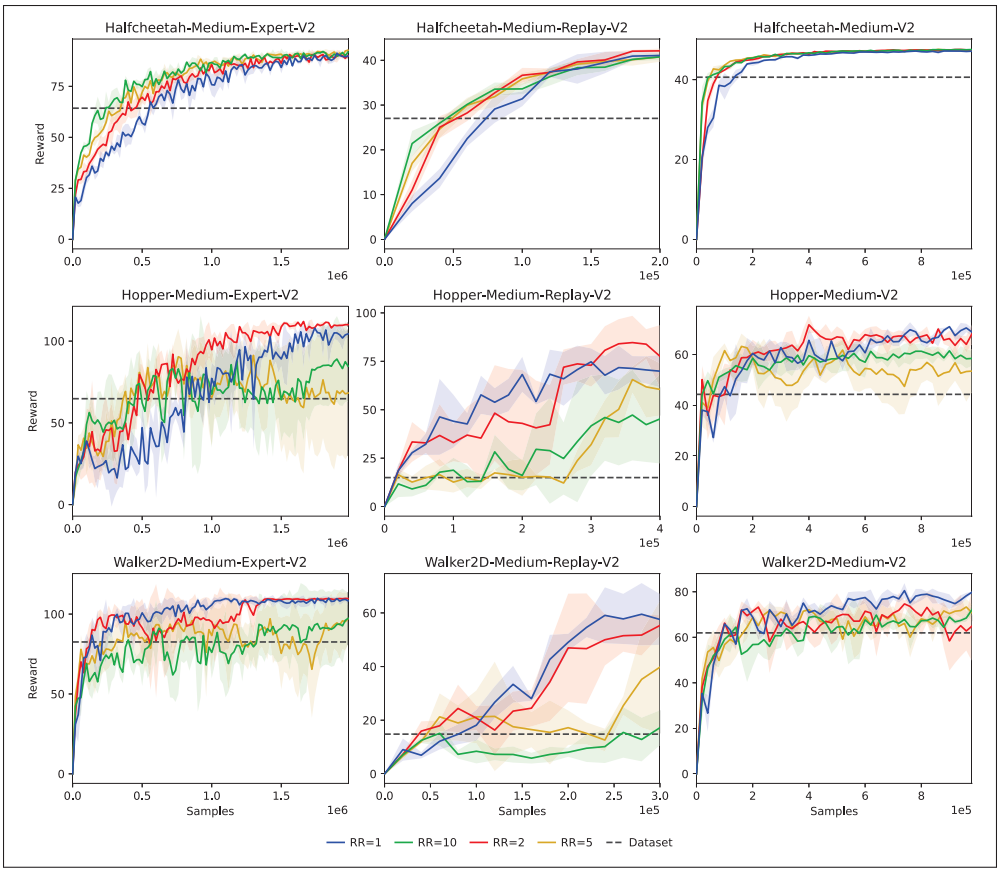


Figure-A I-6 Comparison of replay ratios greater than 1.

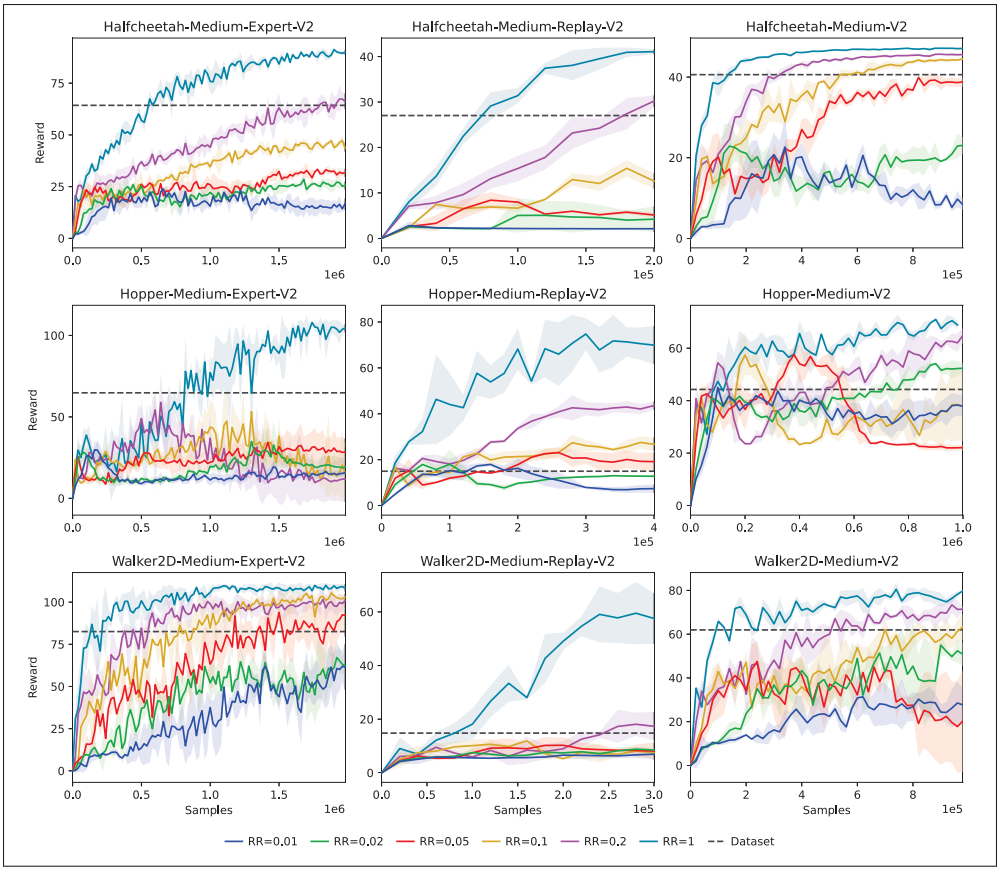


Figure-A I-7 Comparison of replay ratios less than 1.

BIBLIOGRAPHY

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. & Bellemare, M. G. (2021). Deep Reinforcement Learning at the Edge of the Statistical Precipice. *Advances in Neural Information Processing Systems*.

Bellemare, M. G., Dabney, W. & Munos, R. (2017). A distributional perspective on reinforcement learning. *International Conference on Machine Learning*, pp. 449–458.

Bellman, R. (1957). A Markovian Decision Process. *Indiana Univ. Math. J.*, 6, 679–684.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C. et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A. & Mordatch, I. (2021). Decision Transformer: Reinforcement Learning via Sequence Modeling. *arXiv preprint arXiv:2106.01345*.

Dawid, A. P. & Vovk, V. G. (1999). Prequential probability: principles and properties. *Bernoulli*, 5(1), 125–162.

Fu, J., Kumar, A., Nachum, O., Tucker, G. & Levine, S. (2020). D4RL: Datasets for Deep Data-Driven Reinforcement Learning.

Fujimoto, S. & Gu, S. S. (2021). A Minimalist Approach to Offline Reinforcement Learning. *Thirty-Fifth Conference on Neural Information Processing Systems*.

Fujimoto, S., van Hoof, H. & Meger, D. (2018, 10-15 Jul). Addressing Function Approximation Error in Actor-Critic Methods. *Proceedings of the 35th International Conference on Machine Learning*, 80(Proceedings of Machine Learning Research), 1587-1596. Retrieved from: https://proceedings.mlr.press/v80/fujimoto18a.html.

Fujimoto, S., Meger, D. & Precup, D. (2019). Off-Policy Deep Reinforcement Learning without Exploration. *International Conference on Machine Learning*, pp. 2052–2062.

Girshick, R. B., Donahue, J., Darrell, T. & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *Ieee Conference On Computer Vision And Pattern Recognition*. doi: 10.1109/CVPR.2014.81.

Haarnoja, T., Zhou, A., Abbeel, P. & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 80(Proceedings of Machine Learning Research), 1856–1865. Retrieved from: http://proceedings.mlr.press/v80/haarnoja18b.html.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. & Amodei, D. (2020). Scaling Laws for Neural Language Models. *arXiv preprint arXiv: 2001.08361*.

Kostrikov, I., Nair, A. & Levine, S. (2022). Offline Reinforcement Learning with Implicit Q-Learning. *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. Retrieved from: https://openreview.net/forum?id=68n2s9ZJWF8.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25. Retrieved from: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Kumar, A., Zhou, A., Tucker, G. & Levine, S. (2020). Conservative Q-Learning for Offline Reinforcement Learning. *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Retrieved from: https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html.

Levine, S., Kumar, A., Tucker, G. & Fu, J. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv preprint arXiv: Arxiv-2005.01643*.

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D. & Gebru, T. (2018). Model Cards for Model Reporting. *FAT*. doi: 10.1145/3287560.3287596.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv: Arxiv-1312.5602*.

Munos, R. & Szepesvári, C. (2008). Finite-Time Bounds for Fitted Value Iteration. *Journal of Machine Learning Research*, 9(27), 815–857. Retrieved from: http://jmlr.org/papers/v9/munos08a.html.

Nair, A., Dalal, M., Gupta, A. & Levine, S. (2020). Accelerating Online Reinforcement Learning with Offline Datasets. *arXiv preprint arXiv:2006.09359*.

Neumann, G. & Peters, J. (2008). Fitted Q-iteration by Advantage Weighted Regression. *Advances in Neural Information Processing Systems*, 21. Retrieved from: https://proceedings.neurips.cc/paper_files/paper/2008/file/f79921bbae40a577928b76d2fc3edc2a-Paper.pdf.

Ostrovski, G., Castro, P. S. & Dabney, W. (2021). The Difficulty of Passive Learning in Deep Reinforcement Learning. *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 23283-23295. Retrieved from: https://proceedings.neurips.cc/paper/2021/hash/c3e0c62ee91db8dc7382bde7419bb573-Abstract.html.

Peng, X. B., Kumar, A., Zhang, G. & Levine, S. (2019). Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. *CoRR*, abs/1910.00177. Retrieved from: https://arxiv.org/abs/1910.00177.

Poland, J. & Hutter, M. (2005). Asymptotics of Discrete MDL for Online Prediction. *IEEE Transactions on Information Theory*, 51(11), 3780–3795.

Precup, D., Sutton, R. S. & Singh, S. (2000). Eligibility Traces for Off-Policy Policy Evaluation. *International Conference on Machine Learning*. Retrieved from: https://api.semanticscholar.org/CorpusID:1153355.

Rathmanner, S. & Hutter, M. (2011). A Philosophical Treatise of Universal Induction. *Entropy*, 13(6), 1076–1136. doi: 10.3390/e13061076.

Rissanen, J. (1984). Universal coding, information, prediction, and estimation. *IEEE Trans. Inf. Theory*, 30(4), 629–636.

Rissanen, J. (1987). Stochastic complexity. *Journal of the Royal Statistical Society: Series B (Methodological)*, 49(3), 223–239.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3), 210–229.

Schmidhuber, J. (2019). Reinforcement Learning Upside Down: Don't Predict Rewards - Just Map Them to Actions. *arXiv preprint arXiv: 1912.02875*.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. & Silver, D. (2019). Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*. doi: 10.1038/s41586-020-03051-4.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I. & Abbeel, P. (2015). Trust Region Policy Optimization. *arXiv preprint arXiv: 1502.05477*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv: Arxiv-1707.06347*.

Shelhamer, E., Long, J. & Darrell, T. (2014). Fully Convolutional Networks for Semantic Segmentation. *Computer Vision And Pattern Recognition*. doi: 10.1109/CVPR.2015.7298965.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484–489. doi: 10.1038/nature16961.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362, 1140-1144. doi: 10.1126/science.aar6404.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (ed. Second). The MIT Press. Retrieved from: http://incompleteideas.net/book/the-book-2nd.html.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T. & Riedmiller, M. (2018). DeepMind Control Suite. *arXiv preprint arXiv: Arxiv-1801.00690*.

Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. *Commun. ACM*, 38(3), 58–68. doi: 10.1145/203330.203343.

Thomas, P. S., Theocharous, G. & Ghavamzadeh, M. (2015). High confidence off-policy evaluation. *AAAI*. Retrieved from: https://dl.acm.org/doi/10.5555/2888116.2888134.

Vapnik, V. (1991). Principles of risk minimization for learning theory. *Proceedings of the 4th International Conference on Neural Information Processing Systems*, (NIPS'91), 831–838.

Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017). Attention Is All You Need. *NIPS*.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C. & Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782), 350–354. doi: 10.1038/s41586-019-1724-z.

Wang, J., Gao, R. & Zha, H. (2020a). Reliable Off-policy Evaluation for Reinforcement Learning. *arXiv preprint arXiv: Arxiv-2011.04102*.

Wang, Z., Novikov, A., Zolna, K., Springenberg, J. T., Reed, S. E., Shahriari, B., Siegel, N., Merel, J., Gulcehre, C., Heess, N. & Freitas, N. D. (2020b). Critic Regularized Regression. *Neural Information Processing Systems*. Retrieved from: https://arxiv.org/abs/2006.15134v3.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4), 229–256. doi: 10.1007/bf00992696.