# Notes on offline reinforcement learning

Compiled by D. Gueorguiev 1/14/24

## Introductory Notes and Problem Statement

The Reinforcement learning algorithms provide a fundamentally *online* learning paradigm which is one of the biggest obstacles to their adoption in various use cases. The process of reinforcement learning involves iteratively collecting experience by interacting with the environment, typically with the latest learned policy, and the using that experience to improve the policy. In many settings, the online interaction is not really useful or even applicable, because data collection is difficult to guarantee in reliable manner. Furthermore, even in domains where online interaction is feasible, we might still prefer to utilize previously collected data – for example if the domain is with large number of dimensions and complex search space so that effective generalization would require large datasets.

The success of machine learning across a range of problems can be partially attributed to the advent of scalable data-driven learning methods which improve the generalization quality as they are trained with more data. Online reinforcement learning is difficult to reconcile with this conclusion. For low-dimensional and linear models using reinforcement learning usually this is not a significant problem. However, the issue becomes obvious when deep neural nets are incorporated in the reinforcement learning algorithm. So the pertinent question is: can data-driven learning be incorporated with the reinforcement learning objectives such that well-chosen offline data is used to drive the learning process?

Environment

update

(b) off-policy reinforcement learning

buffer

Figure 1a: In online reinforcement learning, the policy is updated with streaming data online collected by the policy itself.

Environment

update

(a) online reinforcement learning

Figure 1b: In off-policy setting, the agent’s experience is appended to the data buffer (also called replay buffer) , and each new policy collects additional data, such that is composed of samples from , and all of this data is used to train an updated new policy .

In contrast, offline reinforcement learning employs a dataset collected by some (potentially unknown) behavior policy . The dataset is collected once, and is not altered during training, which makes it feasible to use large previous collected datasets. The training process does not interact with the MDP at all, and the policy is only deployed after being fully trained.

Figure 1c: In offline reinforcement learning a dataset is collected by some (potentially unknown) behavior policy . The dataset is collected once, and is not altered during training, which makes it feasible to use large previously collected datasets. The training process does not interact with the MDP at all, and the policy is only deployed after being fully trained.

learn

Environment

(c) offline reinforcement learning

buffer

Environment

data collected **once** with **any** policy

training phase

A data-driven offline reinforcement learning as pictured in Figure 1c poses significant algorithmic challenges. Many commonly used reinforcement learning methods can learn from off-policy data, but such methods often cannot learn effectively from entire offline data, without any additional on-policy interaction. High-dimensional and expressive function approximation generally exacerbates this issue, since function approximation leaves the algorithms vulnerable to *distributional shift* (See Appendix) which is one of the central challenges with reinforcement learning.

Despite the mentioned problems, the appeal of fully offline reinforcement learning is considerable: in the same way that fully supervised machine learning methods have enabled data to be turned into generalizable and powerful *pattern recognizers* (e.g. image classifiers, speech recognition engines, etc), offline reinforcement learning methods equipped with powerful function approximation may enable data to be turned into generalizable and powerful decision making engines, effectively allowing anyone with a large enough dataset to turn this dataset into a policy that can optimize a desired utility criterion.

Note: The *Offline Reinforcement Learning* is also known in some of the older literature sources as *Batch Reinforcement Learning* (cf. [4], [5], [6]).

## Offline Reinforcement Learning Problem Statement and Overview

Reinforcement Learning addresses the problem of learning to control a dynamical system, in a general sense. The dynamical system is fully defined by a fully-observed or partially-observed Markov decision process (MDP).

Definition 2.1 (Markov Decision Process).

## Appendix

### What is Distribution Shift?

Empirical Risk Minimization

With the standard fitting paradigm, we try to minimize the error in our estimation using tuples sampled from the problem domain with distributions and respectively. Therefore the fitting problem usually is represented as:

(1)

For expressive enough model and effective learning technique we can assume that will match for . But what about ? Theoretically, the above optimization procedure assures that the error is smalls only if the samples are sampled from the same distribution. (Is this really true?) In reality with the recent advancements in Deep Learning the above-mentioned theoretical limit has been effectively circumvented (see [3]).

Offline RL Q-Learning

Offline RL deals with the problem of acquiring optimized policies for a given problem without having the ability to interact with the environment to collect data according to the policies present up to the current moment. The only source of information is a dataset of previously collected transitions that should be carefully “distilled” or filtered to lead to an optimized policy for the problem at hand. This dataset has been collected by a (usually unknown) policy, called behavior policy, , which can be either human-controlled actions, a rule-based policy, previously learned RL policy or all of the above.

Therefore, in the offline RL framework, we have a learning scheme similar to the previous subsection. Following any -learning scheme, the target values for the Bellman equation update are calculated as follows:

(2)

With Sutton’s notation (2) becomes

where denotes the currently evaluated policy. Later, during the function update, the following objective will be pursued:

(3)

Using Sutton’s notation, which is no less clumsy, (3) becomes:

With model-free offline RL we are forced to use samples from to be able to incorporate the offline reward signal into the Bellman’s update cycle. As discussed in the previous paragraph a good accuracy is expected when

Note that in offline RL we do not want to perform a behavioral cloning but rather to process the data collected with and produce a better optimized policy.

The “Malicious” Actor

As we discussed in the section *Empirical Risk Minimization* usually, we offload that task to the advanced generalization capabilities of Deep Neural Networks. We could garner a large and representative dataset and then the generalization abilities of Deep Neural Networks, utilized inside the framework of RL algorithms will do the generalization hence the error minimization correctly.

Unfortunately, an inherent feature of the learning framework acts against that. Apart from learning the q function, a policy with respect to q function is generated as well, according to:

(4)

Thus, not only the distribution of states is not the one that generated the samples to fit on, but how the states are produced to receive the biggest possible value of the estimated function.

The figure below demonstrates why this process can be quite problematic.

## Figure: Maximization on estimated q (orange line) leads to areas with largest positive error

Maximum

positive

error

True maximum

true

estimated

For ease of visualization and comprehension, we investigate the values of for a specific , i.e. . Also, a scalar action value is assumed. Note that in multi-dimensional action spaces the problem described here is actually exacerbated by the immensity of the search space. The blue line represents the form of the true (unknown) function, and the orange line depicts its current estimation based on the recorded data points (green dots on the Figure). As it is expected, the estimated q is relatively accurate around areas dense in datapoints, while usually tens to miss the true q around areas with little to no information producing equally possible positive or negative errors.

**The Problem**: If we select new points according to a policy similar to that in Eq. (4), then we tend also to select points where the error from the true has the largest positive value, also missing the true maximum points.

In the offline -learning paradigm , the actor chooses the state-action pairs with the largest value of the current estimation of , favoring positively erroneous estimated values. In turn, these unreliable values are used to update the estimation, following equations (2)-(3), further diverging the estimation of from its true function. Given that the q function is not allowed to collect more data around these values, the perfect trap has been set. The actor will fail to solve the task at hand because it enters a “vicious cycle” of updating the q-function with positively erroneous values from areas with little to no actual data (out-of-distribution state-action pairs).

The following figure showcases such an incident. The different colors correspond to different sizes on the collected samples from the mujoco environment HalfCheetah-v2, and the q-learning was the state-of-art SAC.

A graph of a number of numbers and a line graph

Description automatically generated with medium confidenceA graph of a number of steps

Description automatically generated Figure: How well the actor believes it performs Figure: Actual performance

Obviously, the actor believes that a great performance has been achieved (the Figure on the left). However, the actual performance (the Figure on the right) of the learned policy has gotten even worse from the initial random policy. Notice how efficient is this “malicious” actor in finding these out-of-distribution actions. Even for up to 1 million transitions, there are no signs of improvement in the actual performance.

## References

[[1] Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems, Sergey Levine, Aviral Kumar, George Tucker, Justin Fu, 2020](https://github.com/dimitarpg13/self_supervised_learning/blob/main/literature/OfflineReinforcementLearningTutorialReviewLevine2020.pdf)

[2] [The Monster of Distribution Shift in Offline RL and How to Pacify it, Athanasios Kapoutsis, Medium, 2022](https://medium.com/@athanasios.kapoutsis/the-monster-of-distribution-shift-in-offline-rl-and-how-to-pacify-it-4ea9a5db043)

[3] [Generalization in humans and deep neural networks, R. Gierhos et al, Max Planck Institute, 2020](https://github.com/dimitarpg13/self_supervised_learning/blob/main/literature/Generalisation_in_humans_and_deep_neural_networks_Geirhos_2020.pdf)

[4] [Batch Reinforcement Learning, Sascha Lange, Thomas Gabel, Martin Riedmiller, 2012](https://github.com/dimitarpg13/self_supervised_learning/blob/main/literature/Lange_Gabel_EtAl_RL-Book-12.pdf)

[5] [Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method (with annotations), Martin Riedmiller, 2005](https://github.com/dimitarpg13/self_supervised_learning/blob/main/literature/Neural_Fitted_Q_Iteration-First_Experiences_with_a_Data_Efficient_Neural_Reinforcement_Learning_Method_Riedmiller_2005_annotated.pdf)

[6] Tree-Based Batch Mode Reinforcement Learning , Damien Ernst et al, 2005