University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

Offline RL: Approaching Reinforcement Learning in a data-driven manner

Artur Magalhães Rodrigues dos Santos

Final Essay

mac 499 — Capstone Project

Supervisor: Prof. Dr. Denis Deratani Mauá

São Paulo
December 10, 2021

Acknowledgment

Pass on what you have learned.
— Master Yoda

I would like to thank every professor and colleagues that helped through my graduation. Professors José Coelho, Nina Hirata, Carlos Ferreira and Elisabeti Kira for the thoughtful conversations and discussions on the hallways of IME-USP. Professor Denis Maua, who advised me on this work, for the opportunities and knowledge shared.

Also my friends who were always by my side on the good and the bad times, Andrew Ijano Lopes, Pedro Almeida and Eduardo Laurentino. In the library, on IME-USP desks or on C-block, they always provided help and support, and shared many cups of coffee.

Additionally, my beloved family, who always encouraged me to continue my studies and not give up. And my sweetheart, Natalia Morita, who is always by my side, giving me support and cheering for my accomplishments.

Resumo

Artur Magalhães Rodrigues dos Santos. **Aprendizado por reforço offline: Abordando Aprendizado por reforço orientado a dados**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Este trabalho consiste em uma visão geral do Aprendizado por Reforço *Offline*, área de pesquisa que faz parte de Aprendizado por Reforço. Abordamos técnicas modernas de Aprendizado por Reforço, que usam apenas dados *online*, até algoritmos completamente *offlines*. Recentemente, a área tem ganhado atenção devido ao seu uso de dados estáticos. Apresentaremos, discutiremos e compararemos os algoritmos de Aprendizado por Reforço *offline*, fornecendo uma revisão da literatura para futuras discussões.

Palavras-chave: Aprendizado por reforço. Aprendizado por reforço offline. Aprendizado de Máquina. Inteligência Artificial.

Abstract

Artur Magalhães Rodrigues dos Santos. **Offline RL: Approaching Reinforcement Learning in a data-driven manner**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

This work consists in an overview on Offline Reinforcement Learning, research area which is part of Reinforcement Learning. We go through modern RL approaches, which only use online data, up to full offline RL algorithms. Recently, the field has gained attention due to its usage of static data. We present, discuss and compare the Offline RL algorithms, providing a literature review for future discussions.

Keywords: Offline Reinforcement Learning. Reinforcement Learning. Machine Learning. Artificial Intelligence.

List of abbreviations

RL	Reinforcement Learning
IME	Institute of Mathematics and Statistics
USP	University of São Paulo
MC	Monte Carlo
NLP	Natural Language Processing
MDP	Markov Decision Process
KL	Kullback–Leibler divergence
DDPG	Deep Deterministic Policy Gradient

List of symbols

π	policy
π_θ	policy parametrized by a set of weights θ
π_β	behavior policy
D	dataset

Contents

Chapter 1

Introduction

Reinforcement Learning, as defined by R.S. Sutton and Barto, 2018, is "*learning what to do—how to map situations to actions—so as to maximize a numerical reward signal*". The framework consists of an agent interacting with an environment and receiving some kind of reward. The agent must discover and learn about the environment on its own, choosing between the available actions. Basically, as described before, the RL framework contains: an environment, an reward, actions and observations.

Generally, the environment is where the agent act upon. The agent, with its set of available actions - which may vary depending on the problem been approached - decides which one to choose, given what is has learned from those previous interactions. It collects observations that describe what the environment looks like after taking an action (and going to a new state), which them are used on the learning process.

The reward is one of the most important parts of the learning loop, due to its effect on the agent behavior. It is the base metric to inform the agent if it has performed well or badly given it took an specific action on the current state. An environment modeled with rewards that don't match the expected agent behavior may lead to the agent having a poor performance. For example, rewarding a agent when it loses a match or a challenge would incentivize it to do so, an odd scenario when we actually want it to perform better - more score, goals etc.

This research area has grown over time, specially after the emergence of deep learning techniques, which use deep neural networks for function approximation (Mnih *et al.*, 2015). Many works have established remarkable results in tasks which humans were thought to be unbeatable, such as Go, Starcraft, and many more. Besides its applications related to gaming , reinforcement learning is used in robotics (Levine, Finn, *et al.*, 2016), recommendation systems (Covington *et al.*, 2016) and more.

Algorithms may vary between being on-policy and off-policy, which information they use to estimate values - state-action pairs or just the state -, on how they balance exploration and exploitation - which is chosen between acting greedily or allowing short term losses for the sake of having new interactions - and many more.

Off-policy algorithms (we will discuss them on Chapter 2, section On-policy and

Off-policy), which act without following its target policy, can in theory use a collection of transitions, stored in a dataset, to learn. But researchers show that empirically these algorithms haven't performed well (Kumar, Fu, *et al.*, 2019). Given the restrictions imposed by the fixed dataset, and without the ability to correct - by interacting with the environment - sub-optimal behavior, off-policy algorithms have a degraded performance, as we will see in Offline Reinforcement Learning.

Offline Reinforcement Learning, also known as Batch Reinforcement Learning (Lange S., 2012), is a data-driven approach to the reinforcement learning problem setting. Differently from the standard reinforcement learning framework, Offline RL makes use of a fixed dataset containing observations of states, actions, and rewards, derived from previous interactions, random interactions, human demonstrations, or demonstrations from related tasks and an agent interacts with this data to gain knowledge on the task at hand.

It basically removes the online interactions procedure, where the agent actively acts on the environment to verify or assert its assumptions - was this a good state or action? - and makes use of only what is contained on the dataset. One may find easier to think of it as in the supervised learning approach, where we want to infer or predict values based on a previously collected dataset.

In many real world problems, collecting new data can be cumbersome, both in terms of cost and required labor. Some particular tasks require human labor to produce data, and have limitations on experimentation time, like training self-driving cars (Sallab *et al.*, 2017) and robotic tasks (OpenAI *et al.*, 2019). Also, some settings may require restrictions on safety, such as healthcare applications (Gottesman *et al.*, 2018), which testing with patients may not be viable.

Offline Reinforcement Learning is notably interesting for approaching these scenarios. Furthermore, other research areas, such as *NLP* (Chelba *et al.*, 2014) and Computer Vision (Deng *et al.*, 2009), rely on models that were previously trained on huge datasets that are refined for solving particular tasks. Hereupon, Offline RL relates to these other research area and bring those ideas of using trained models and only refining them for different tasks into RL.

The main challenges posed by Offline RL are mainly on how to use the data properly, that is, learn from the dataset but being able to generalize the behavior outside the dataset. In most cases, the dataset contains data that may be not reliable (partially complete interactions, sub-optimal behavior), posing a more challenging situation to the learning problem. We will discuss further how these problems emerge and are approached on previous research.

Additionally, most Offline RL algorithms don't obtain much improvement when learning from an offline dataset and fine-tuning with an online environment (Levine, Kumar, *et al.*, 2020, Nair *et al.*, 2021). Problems related to fitting a behavior model, which is modeled after the offline data and its imperfections, may occur due to the algorithms being over conservative or optimistic. Thus, when the behavior model is inaccurate, it may become too conservative or too optimistic over new data and it isn't able to improve much over online interactions on the environment, leading to poor improvements and slow learning

(Nair *et al.*, 2021).

Some established works bridge this gap between Offline RL - which tends to be over conservative but has advantages on data usage and collection - and RL - which is able to learn on a online setting balancing exploration and exploitation on the interactions it takes - combining these 2 approaches to accelerate the learning process. Others may approach this problem being fully offline, only relying on what's on the dataset. We will discuss both of them in Offline Reinforcement Learning.

In our work, we will discuss and dive deep on Offline RL techniques in a broader range, explaining the challenges and state-of-the-art approaches. We will walk through the main differences between Online and Offline Reinforcement Learning, and why it is an interesting topic and research area. We will give a quick overview on the fundamentals of Reinforcement Learning and on some established algorithm architectures.

The objective is to provide students and researches a first step on understanding Offline Reinforcement Learning, with both theoretical discussions and showing some state-of-the-art algorithms.

Chapter 2

Preliminaries

Online Reinforcement Learning (R.S. Sutton and Barto, 2018), or just Reinforcement Learning, is a learning framework where an agent interacts with an environment, looking to maximize its expected future (discounted) reward, based on which actions and decisions it took.

To exemplify this interactions, lets proceed with a simple example: a puppy discovering the world around it. As it progresses through its life, it gets better on deciding what's good - maybe a crunchy snack - and what's bad - eating the furniture around the house.

In the Reinforcement Learning framework, the agent is the puppy, the environment is where it lives and runs around, it decides what to do by taking actions, and it learns a good behavior by the "rewards" collected from its interaction with the environment.

Reinforcement Learning has achieved remarkable results, specially on the last couple years, along with the advancements on Deep Learning (LeCun *et al.*, 2015). The development of new techniques and algorithms, capable of beating the best players on games where humans thought to be the best (Silver *et al.*, 2017), providing insights and solutions to problems in biology, engineering, robotics, health, and many others, has shown the potential of this research area.

The field of Deep Learning, which makes use of Deep Neural Networks, enabled the construction of better agents, more skilled and perceptive on its surroundings. It also leveraged using high dimensional data, such as images, on the Reinforcement Learning process. Deep Neural Networks are fundamental piece on building better agents because they are more robust on approximating functions than shallow networks (with few hidden layers) (Mhaskar *et al.*, 2017).

Although the Online Reinforcement Learning setting remains the same, with its classic interaction loop, how the agent learns varies significantly across different techniques. An agent may learn a value function, trying to quantify the quality of a state or state-action pair. It may try to learn a policy directly, using policy gradient methods, or combining these 2 approaches, with Actor Critic methods. We will investigate them in Recent research.

Each one of them have differences on how they update its values, on following or not its current behavior policy, and many more. Another difference worth mentioning is

between techniques which are based on learning a model from the environment, through a parametrized transition model, and those who are not. These are called model-based and model-free methods, respectively. We will discuss their differences further, as they are also important for our Offline Reinforcement Learning scenario.

2.1 The Reinforcement Learning problem

We will proceed defining the notation and necessary terms. Along this work, they will be needed for understanding the learning process. This definition is broadly known, and it follows Levine, Kumar, *et al.*, 2020 and R.S. Sutton and Barto, 2018. The Reinforcement Learning problem can be defined as a Markov Decision Process (MDP):

Given a set of states S (state $s \in S$), a set of actions A (action $a \in A$), a reward function $r(s, a)$ where $r : SxA \longrightarrow R$, a scalar discount factor $\gamma \in (0, 1]$ and a conditional probability distribution $p(s_{t+1}|s_t, a_t)$ over a state s_t and action a_t, we define a Markov Decision Process (MDP) as $M = (S, A, p, r, \gamma)$. The case where we have a partially observable MDP is omitted, but it follows in the same way.

To exemplify, you may think of some classic Gym (Brockman *et al.*, 2016) environments. *Cartpole*, an environment where a cart needs to balance a pole over a frictionless track has a state s defined as the cart's position and velocity, and pole angle and angular velocity, a tuple with 4 values. The actions are discrete, going left or right. The reward value is 1 for every step - this promotes the agent's behavior to balance the pole more timesteps.

With our MDP defined, our main goal is to learn a policy π, more precisely, $\pi(a|s)$ - it tells us what action to take on a given state, that maximizes the cumulative discounted return:

$$V_\pi(s) = E_{a\sim\pi}[\sum_{t=0}^{H} \gamma^t r(s_t, a_t)] \tag{2.1}$$

Under policy π when starting from a state s, where H may be finite or infinite. This defines our **Value Function**. Similarly, our cumulative discounted reward may also be over a state-action pair, so we define it as:

$$Q_\pi(s, a) = E_{a\sim\pi}[\sum_{t=0}^{H} \gamma^t r(s_t, a_t)] \tag{2.2}$$

Defining our **Action-value function**, also known as Q-function.

Following Levine, Kumar, *et al.*, 2020, we define the overall state visitation frequency averaged over time steps $d^\pi(s)$ and $d_t^\pi(s_t)$ the state visitation frequency at time step t. The notation related to Offline RL will be elaborated afterwards, on Definitions section.

In other words, the traditional online problem consists of the iterative approach of an agent interacting with the environment, performing an action on it, and collecting what happened afterwards: what are the rewards? what's my next state? Then, it can use this information to learn - for example, estimating how good a state or state-action tuple was - and perform better on a possible next decision. It reasons about what's better to do next.

We will see that the offline setting, although similar, differs in the interaction loop: actually, there's no interaction, and we are left only with a dataset D of transition tuples. We will investigate further on Offline Reinforcement Learning.

2.2 On-policy and Off-policy

As mentioned before, reinforcement learning methods may vary on how they update their respective estimate of state (or state-action) values which changes the agent's behavior. For the difference between on-policy and off-policy, is important to recall what the agent's behavior policy is. It is the policy the agent uses to interact with the environment, or in other words, the one it uses to choose actions over states.

On-policy approaches update its values based on the current behavior policy. The agent interacts with the environment, choosing an action a_t and getting to a new state s_{t+1} with a reward r_t, and uses them to update its current policy π.

It improves on the same policy it is following, so the behavior policy is the same as the policy used for taking actions. SARSA (R.S. Sutton and Barto, 2018), PPO (Schulman *et al.*, 2017) are examples of on-policy algorithms.

By contrast, **off-policy** methods don't necessarily update its values based on current behavior policy. This means a off-policy algorithm may collect data and only use it for learning afterwards. Q-learning (Watkins and Dayan, 1992), DDPG (Lillicrap *et al.*, 2019) and SAC (Haarnoja *et al.*, 2018) are examples of off-policy algorithms.

Generally, this data is stored on a replay buffer, a common data structure used to store collected transitions (more about it on DDPG). Or it may improve over another policy, which is the case of taking *max* - greedy policy - on Q-learning (equation on Q-learning), making it a off-policy method. As we will see, off-policy methods can indeed be used for offline training, but have noticeable drawbacks (Kumar, Fu, *et al.*, 2019).

It is important to delimit the difference between each of them as in the offline scenario they have peculiarities and special considerations, due to their nature and how the Offline RL problem is formulated. In the Offline Reinforcement Learning section we will investigate them in more detail.

2.3 Model-free and Model-based

Another difference between Reinforcement Learning techniques is if they learn a model of the environment or not. In the Reinforcement Learning setting, we don't have all the

information available for the MDP, specially the reward function R and the transition function $T(s_{t+1}|s_t, a_t)$.

Model-based methods try to model these functions, the environment dynamics. With this learned model, it can make predictions on the expected next state and reward and query the model. Dyna (Richard S. Sutton, 1990) is an example of model-based algorithm.

On the other hand, **model-free** methods don't do so. They derive a policy by learning from experience, without using predictions on the upcoming states and rewards, relying on their own interaction with the environment. Q-learning (Watkins and Dayan, 1992) is an example of model-free RL algorithms, as they rely on interacting and improving upon it.

Model-based and model-free are not to be confused to using function approximators or not. A reinforcement learning algorithm which uses neural networks for function approximation, as we will see further, may be model-free or not.

In Offline Reinforcement Learning, we will discuss different approaches on offline reinforcement learning with the perspective of model-free and model-based methods.

2.4 Recent research

We will now discuss about recent research on Reinforcement Learning, going through Q-learning, policy gradient and Actor Critic concepts, and presenting Deep Deterministic Policy Gradient (Lillicrap *et al.*, 2019), an off-policy model-free reinforcement learning algorithm.

2.4.1 Q-learning

Q-learning (Watkins and Dayan, 1992) is one of the most traditional reinforcement learning methods. It consists of learning Q-values through a Q-function, a value function based on state-action pairs, therefore, it is a value function method. Also, because it doesn't rely on modeling the environment transition function $T(s_{t+1}|s_t, a_t)$, it is a model-free algorithm.

Given a policy π, the classical Q-function consists of:

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma * E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t), a_{t+1} \sim \pi(a_{t+1}|s_{t+1})}[Q(s_{t+1}, a_{t+1})] \tag{2.3}$$

In summary, it says that our state-action estimate, which is what helps our agent to learn, is a result of the present reward $r(s_t, a_t)$ and a expected discounted cumulative sum of rewards following the possible next steps. It can be derived from the value based function $V(s)$ for a given state s (R.S. Sutton and Barto, 2018).

Given that we want to maximize our return, for the update equation, we interpret the expectation term as a *max* over the future:

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma * \sum(\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})) \tag{2.4}$$

And we update our estimates on state-action pair with the temporal difference update, approaching Q^*, the optimal Q-value for a given state-action pair:

$$Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha(r(s_t, a_t) + \gamma * max(Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t)) \tag{2.5}$$

For update in Equation 2.5, Q-learning is off-policy, as it uses the max function, and not actually learning on the current policy it is following.

The Q-learning algorithm pseudocode is presented below:

Algorithm 1 Q-learning pseudocode

$\alpha \in (0, 1]$ and $\epsilon > 0$
Initialize $Q(s, a)$
for episode e = 1, 2, ..., E **do**
  for t = 0, 1, ... **do**
    Choose action a from s using Q (e.g. ϵ-greedy)
    Take a and get (s', r)
    Update $Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_a Q(s', a) - Q(s, a))$
    $s = s'$
  end for
end for

2.4.2 Policy gradient

Policy gradient methods, introduced in Richard S Sutton *et al.*, 2000, rely on the idea of learning a policy π directly, meaning we adjust the policy, now a parametrized function π_θ, according to the direction - gradient of the policy - that makes it better - results in higher rewards. This direction is taken using the gradients regarding this policy, commonly, using gradient ascent.

The objective function defined for policy gradient, opening the expectation term in regard of $\pi(s|a)$:

$$\begin{aligned} J(\theta) &= \sum_{s \in S} d^{\pi}(s) V^{\pi}(s) \\ &= \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \pi_\theta(a|s) Q^{\pi}(s, a) \end{aligned}$$

Vanilla policy gradient, also know an REINFORCE (Monte Carlo Policy Gradient) (Richard S Sutton *et al.*, 2000), the most simple policy gradient estimation, first calculates an estimate after timestep t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.6}$$

And given that Q can be stated in terms of:

$$Q(s_t, a_t) = E_\pi[G_t|s_t, a_t] \tag{2.7}$$

Thus calculates the gradient of the objective function, in terms of the expectation:

$$\nabla_\theta J(\theta) = E_\pi[G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)] \tag{2.8}$$

Differently from value based RL, policy gradient methods don't need to learn a value function, which may be helpful, specially when modeling a complex value function is expensive and cumbersome. One remarkable result from policy gradients is being able to learn stochastic policies - probabilities over actions -, expanding on what kinds of environments an agent is able to learn.

More recent methods also work efficiently with high dimensional and continuous action spaces. Policy gradient methods rely on the Policy Gradient Theorem (Richard Sutton *et al.*, 2000) which states:

Theorem 2.4.1 (Policy Gradient Theorem)**.** *For any differentiable policy $\pi_\theta(s, a)$, for any policy objective function J, the policy gradient is*

$$\nabla_\theta J(\theta) = E_\pi[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$$

Nonetheless, when done naively, problems like converging to local optima and inefficient policy evaluation, due to high variance, may arise. Even when collecting a full trajectory before updating the policy weights, the difficulty of knowing which action provided more reward and thus is better persists.

The vanilla policy gradient algorithm is presented below:

Algorithm 2 Vanilla Policy Gradient pseudocode

```
Policy parameters θ
for t = 1, ..., T do
    Collect s_1, a_1, s_2, ..., s_T, a_T
end for
for t = 1, 2, ..., T do
    Estimate G_t = Σ_{k=0}^{∞} γ^k R_{t+k+1}
    Update θ = θ + α γ^t G_t ∇_θ ln π_θ(a_t|s_t)
end for
```

2.4.3 Actor Critic

Constructing upon Policy Gradient, we reach Actor Critic methods. It introduces the idea of a Critic to estimate the action-value function $Q_w \sim Q^{\pi_\theta}(s, a)$, one step further from Policy Gradient methods as it also learns a explicit value function with a function

approximator. Thereby, it assists on the policy update, reducing the gradient variance when comparing to a simple policy gradient algorithm.

Actor Critic methods rely on having 2 function approximators: one for the Actor, one for the Critic. In recent research, they are usually implemented as Deep Neural Networks (Lillicrap *et al.*, 2019, Haarnoja *et al.*, 2018).

As the name suggests, the Actor is responsible for taking actions based on a given state s and is modeled as $\pi_\theta(a|s)$. The Critic evaluates those states, solving the policy evaluation problem. It is usually modeled as $Q_\phi(a, s)$. It follows the approximate policy gradient defined by:

$$\nabla_\theta J(\theta) \sim E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q_\phi(s, a)] \tag{2.9}$$

$$\delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) Q_\phi(s, a) \tag{2.10}$$

We saw that Policy Gradient can have high variance, specially for Vanilla Policy Gradient, and Actor Critic handles it by using an advantage function, defined as:

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s, a) \tag{2.11}$$

The reasoning behind it is to measure how much better (more reward) than usual we can be by taking a specific action. It is the difference between the state-action and state value functions. It builds up on the idea of subtracting a baseline function $B(s)$ from the policy gradient:

$$\begin{aligned} E_{\pi_\theta}[\nabla\theta \log \pi_\theta(s, a) B(s)] &= \sum_{s \in S} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s, a) B(s) \\ &= \sum_{s \in S} d^{\pi_\theta} B(s) \nabla_\theta \sum_{a \in A} \pi_\theta(s, a) \\ &= 0 \end{aligned} \tag{2.12}$$

The baseline function has the benefit of reducing variance without changing expectation. Recall that we don't need to calculate both Q and V because the temporal difference error (as in Equation 2.11) is an unbiased estimate of the advantage function A and can be used to derive it.

The vanilla Actor Critic algorithm pseudocode is presented below:

Algorithm 3 Vanilla Actor Critic pseudocode

Critic Q_θ, Actor μ_ϕ
Replay buffer D
for t = 1, ..., T **do**
  Get $a_t = \mu_\phi(s_t)$
  Execute a_t and collect transition (s, a, s', r)
  Update (Actor) policy parameters $\phi = \phi + \alpha Q_\theta(s, a) \nabla_\phi \ln \mu_\phi(a|s)$
  Compute temporal difference error: $\delta_t = r_t + \gamma Q'_\theta(s_{t+1}, \mu'_\phi(s_{t+1})) - Q_\theta(s, a)$
  Update Critic: $\theta = \theta + \alpha \delta_t \nabla_\theta Q_\theta(s, a)$
end for

2.4.4 DDPG

A established Deep RL method, DDPG (Lillicrap *et al.*, 2019) combines using Q-learning along with policy gradients, as it is an Actor Critic method. It also derives some concepts from DQN (Mnih *et al.*, 2015), using replay buffers and target networks. It is an off-policy and model-free algorithm. We will dive deep on it.

For Deep Deterministic Policy Gradient, the Actor network $\pi_\mu(a_t|s_t)$ receives the state s_t as input and outputs an action a_t. The Critic network $Q_\theta(s_t, a_t)$ receives both states and actions (which the Actor outputs) and evaluates them with a Q-value as the output.

It seems straightforward to learn with these architecture, but it is necessary to recall some important aspects on why DDPG works. The usage of replay buffers and target networks, advancements elaborated on DQN paper, are essential contributions that stabilize the learning process.

Basically, the replay buffer is a structure where transitions are stored and can be queried by the agent afterwards. The target networks are function approximators and their weights are copies of the weights of the Actor and the Critic after n time steps.

Replay buffer

First, the replay buffer. It is used for taking random samples of observations (also called transitions), which were collected by the agent, and it is applied during learning. At each time step, a mini batch of transitions is collected and stored on the replay buffer, to be used by the Actor and Critic. Its objective is to break temporal correlation between training episodes.

If we used sequential transitions, we would introduce more variance on approximating the Q-value. The temporal difference equation compounds over the introduced variance, leading to poor performance. When the replay buffer is full, the oldest samples are discarded. It can be compared to a cache from the agent transitions.

Given the replay buffer, which contains random samples from transitions previously experienced by the agent, DDPG is considered an off-policy algorithm, as it may use experiences from an outdated policy. This isn't a concern since the Bellman equation update doesn't care which transitions - at specific time steps - are under evaluation.

Target networks

Regarding target networks, they provide more stability to the algorithm. During the calculation of the temporal difference error, directly updating the Actor and Critic weights (μ and θ) may lead to divergence. This idea comes from Mnih *et al.*, 2015, where the authors suggest that updating $Q(s_t, a_t)$ would often increase $Q(s_{t+1}, a)$, thus leading to oscillations and divergence on the policy.

Using a previous set of parameters - weights from target networks, which have a delay to the actual Actor and Critic networks, reduces this problem. One noticeable difference for DDPG is that it does soft-updates, meaning the target network weights are updated as:

$$\theta' = \tau\theta + (1 - \tau)\theta', \tau << 1$$

The authors recall that the slow change provided by the soft-update increases stability. In addition, although the learning speed is reduced, due to the additional overhead of having 2 more neural networks for the target networks, it further improves stability and outweighs the negative speed effects.

The Actor update relates to the policy gradient update, where we apply the gradient to the expected return with respect to the Actor parameters:

$$E_{s_t \sim \rho^\beta}[\nabla_a Q_\theta(s, a)_{s=s_t, a=\mu(s_t)} \nabla_\phi \mu(s)|_{s=s_t}]$$

And the Critic update relates to minimizing the loss:

$$L = \frac{1}{N} \sum_t (y_t - Q_\theta(s, a)|_{s=s_t, a=\mu(s_t)})^2$$
$$y_t = r_t + \gamma Q'_\theta(s_{t+1}, \mu'_\phi(s_{t+1}))$$

DDPG also introduces batch normalization (Ioffe and Szegedy, 2015) to its neural networks. This technique, common for deep learning applications, normalizes each dimension across the values from a mini batch, ensuring unit mean and variance.

In reinforcement learning, where different signals with distinct scales come from an environment, its an effective approach. Besides helping on convergence, the authors suggest that implementing batch normalization on the Actor and Critic layers allowed the DDPG agent to learn more effectively across different tasks.

Furthermore, for dealing with exploration, they introduced an exploration policy, the original policy added by a noise sampled from a noise process N. The authors used the *Ornstein-Uhlenbeck* process to generate temporally correlated exploration, leading to:

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + N$$

The Deep Deterministic Policy Gradient pseudocode is presented below:

Algorithm 4 DDPG pseudocode

Critic Q_θ, Actor μ_ϕ
Target networks Q' and μ' with weights $\theta' \leftarrow \theta$, $\phi' \leftarrow \phi$
Replay buffer D
for episode e = 1, 2, ..., M **do**
    Random process N for exploration
    for t = 1, ..., T **do**
        Get $a_t = \mu_\phi(s_t) + N_t$
        Execute a_t and store transition (s, a, s', r) on D
        Sample D for n transitions
        Critic target: $y_t = r_t + \gamma Q'_\theta(s_{t+1}, \mu'_\phi(s_{t+1}))$
        Update Critic: $\theta = \min \frac{1}{N} \sum_t (y_t - Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)})^2$
        Update Actor: $E_{s_t \sim \rho^\beta}[\nabla_a Q_\theta(s, a)_{s=s_t, a=\mu(s_t)} \nabla_\phi \mu(s)|_{s=s_t}]$
        Update Q': $\theta' = \tau\theta + (1 - \tau)\theta'$
        Update μ': $\phi' = \tau\phi + (1 - \tau)\phi'$
    end for
end for

Chapter 3

Offline Reinforcement Learning

As introduced above in Introduction, the Offline Reinforcement Learning (Offline RL) problem relates to using a fixed dataset D for training an agent (it may also be referred as Batch Reinforcement Learning). This dataset D contains the transitions $(s_t^i, a_t^i, s_t^{i+1}, r_t)$ that our agent will use to learn a policy.

This research area has been gaining attention on recent years. The idea of using huge collected datasets for training deep neural networks leveraged Artificial Intelligence and Machine Learning fields, as Computer Vision (Deng *et al.*, 2009), Natural Language Processing (Chelba *et al.*, 2014) and others.

The iteration process of an online reinforcement learning algorithm focuses on collecting new experiences at every step, so the benefits that comes from using a large dataset is missing. This is where Offline Reinforcement Learning comes in.

Besides leveraging datasets, some areas have domains which are costly to interact with, for example, robotics (Levine, Finn, *et al.*, 2016). For training reinforcement learning algorithms, one must rely on simulators, training with many steps, and usually make use of deep neural networks, which can incur in extra costs and time.

Another issue is that some domains are sensitive or even dangerous to permit errors, for example, healthcare (Gottesman *et al.*, 2018). It isn't possible to train an agent that mistakenly gives the wrong treatment to a patient, so again, researches must rely on simulators or other kinds of environments that mitigate this risk. Offline RL allows the creation of a dataset with heterogeneous transitions - expert, sub-optimal and random transitions - which can be used for training an agent, without the risks of online interactions.

The main challenge with this approach is that when we learn from a dataset, which follows a distribution, our agent will be evaluated when interacting with a test dataset, or even in a online environment, with a distribution of state visitation which may be quite different from the one we observed.

More formally, the learning agent on Offline RL needs, from just a subset of experiences collected from a MDP M, to be capable of learning a policy π with the best performance - largest expected cumulative reward - when interacting with M. This is called distributional shift, and we will discuss it later when discussing the algorithms.

Other particular problems arise from learning from a fixed dataset. First, we can't correct our behavior with exploration, relying only on dataset D. So if our dataset D is flawed, meaning it doesn't contain sufficient transitions, they don't contain high reward regions or are concentrated in just one specific region of the environment, we may perform badly (Kumar, Fu, *et al.*, 2019, Wu *et al.*, 2019, Kumar, Zhou, *et al.*, 2020).

This is an important point for Offline RL, as we need to be able to generalize for what's outside D and also because our dataset may contain transitions that are sub-optimal. If all transitions in D were already the best ones, probably it would make more sense to just apply imitation learning.

Off-policy online reinforcement learning algorithms have been used with fixed data as a way of approaching offline training. Particularly, standard Actor Critic and Q-learning methods can learn from data collect outside the agent's behavior policy π_β, but are more impacted by distribution shift than strictly offline methods (Kumar, Fu, *et al.*, 2019, Fujimoto *et al.*, 2019).

Besides distributional shift, these methods, when used in the offline setting, tend to overestimate the values learned, and perform poorly on real scenarios. Overestimation is also a particular problem for the offline setting, where we don't have the benefit of interacting with the environment to correct our estimation (Wu *et al.*, 2019).

There are different approaches for correcting overestimation. We may apply the constraints on the policy, leading to policy constraint methods. Also, we may approach this problem constraining directly on the learned value function, known as policy penalty methods. This nomenclature comes from Levine, Kumar, *et al.*, 2020.

Finally, we may try to estimate the uncertainty of the learned values, known as uncertainty estimation methods. We will also discuss the differences between model-free and model-based algorithms when dealing with offline data.

Regarding implementation and code, Seno and Imai, 2021 provide a remarkable Python interface, based on Pytorch (Paszke *et al.*, 2019), with most state-of-the-art algorithms for Offline Reinforcement Learning - and with the ones we will discuss below. It is versatile and easy to use, and the implementation relies on the algorithm's original paper. It is worth considering it if one plan to use Offline RL.

3.1 Definitions

As stated before, the main goal of reinforcement learning is to learn a policy that maximizes expected cumulative discounted reward in a MDP, defined by $M = (S, A, p, r, \gamma)$. For the offline reinforcement learning setting, it will be useful to have some more definitions.

The behavior policy is defined by $\pi_\beta(a|s)$, D is a dataset with transitions $(s_t, a_t, s_{t+1}, r_t)_i$, $d^\pi_\beta(s)$ a discounted marginal state-distribution of $\pi_\beta(a|s)$ and $d^{\pi_\beta}_t(s_t)$ the marginal over timestep t.

3.2 Model-free approach

Model-free algorithms, specially approximate dynamic programming methods try to learn a value or state-action function which is then used to find the optimal policy (or estimate the gradient of the expected return, in Actor Critic). As we mentioned before, they can be used in the offline scenario, but with many flaws, distributional shift being a noticeable one (Kumar, Fu, *et al.*, 2019, Wu *et al.*, 2019).

It affects the algorithm in different ways. First, regarding out-of-distribution states. The state distribution $d^{\pi}(s)$ differs from the distribution of the dataset D d^{π_β}, and when training, either using policy gradients - Actor Critic - or using a greedy policy - Q-learning -, the learned policy can lead to overestimated values for states $s \sim d^{\pi_\beta}$.

Other possibility is to have action distributional shift, a result of out-of-distribution actions, as the target values for the Bellman backups depend on action $a_{t+1} \sim \pi(a_{t+1}|s_{t+1})$. When we compute the target values, for example, Q, we depend on the action available to us at training time. Thus, we may have overestimated values for actions outside our training data (Levine, Kumar, *et al.*, 2020).

Most online reinforcement learning algorithms can correct their errors interacting directly with the environment, which is not the case for offline RL. So its important to address those issues to have an effective offline algorithm.

Constraint types

Following Levine, Kumar, *et al.*, 2020, policy constraint methods are those which apply some restriction, either implicitly or explicitly, ensuring the distribution over actions for $\pi(a'|s')$ is close - closeness usually defined by a divergence measure - to the behavior distribution $\pi_\beta(a'|s')$. The intention is to allow the policy to improve over the behavior policy, but reducing the deviations and errors due to distributional shift, by keeping $\pi(a'|s')$ sufficiently close to $\pi_\beta(a'|s')$.

For uncertainty estimation methods, the idea is to estimate the uncertainty of value function, via a uncertainty measure, to induce conservative target values, thus reducing overestimation on out-of-distribution actions.

The algorithms we will discuss reside on the policy constraint category, which can be further categorized in 4 types: (1) explicit f-divergence constraints, (2) implicit f-divergence constraints, (3) integral probability metric and (4) policy penalty.

The f-divergence constrains relate to a divergence measure, usually KL or total variation distance, which keeps the policy π close to π_β, given an ϵ: $D(\pi, \pi_\beta) \leq \epsilon$. The restriction can be applied explicitly, adding the constraint to the update of the policy, or implicitly, by constructing a update that considers this divergence measure. Policy penalty relates to adding a penalty term either on the target Q-value or in the reward function, with a penalty term $\alpha D(\pi(.|s)|\pi_\beta(.|s))$, in the reward function case leading $\hat{r} = r(s,a) - \alpha D(\pi(.|s)|\pi_\beta(.|s))$, and in the target case:

$$\hat{Q}^{\pi}_{k+1} = argmin_Q E_{(s,a,s')\sim D}[(Q(s,a) - (r(s,a) + \gamma(E_{a'\sim\pi_k(a'|s')}[\hat{Q}^{\pi}_k(s',a')] - \alpha D(\pi(.|s)|\pi_\beta(.|s)))))^2] \quad (3.1)$$

$$\pi_{k+1} = argmax_\pi E_{s\sim D}[E_{a\sim\pi(a|s)}[\hat{Q}^{\pi}_{k+1}] - \alpha D(\pi(.|s)|\pi_\beta(.|s))] \quad (3.2)$$

We will not go into detail for integral probability metric, but in summary, it is another possibility of applying a policy constraint with a divergence metric but having other theoretical results that have some guarantees and properties.

f-divergences have the particular advantage of being simpler to implement when used as policy penalty, because it needs just an alteration on the reward function with the penalty term. Nonetheless, as discussed in Levine, Kumar, *et al.*, 2020, Kumar, Fu, *et al.*, 2019 and Laroche *et al.*, 2019 show that restrictions on the support of the learned and behavior policy may lead to better expected agent behavior, exemplified in a 1D line world environment, with an uniformly random distribution.

Distribution constrains, such as the KL-divergence, are unable to find an optimal policy, producing a highly stochastic and sub-optimal policy. Support constrains have the benefit of preventing out-of-distribution actions with more strong theoretical and empirical results (Kumar, Fu, *et al.*, 2019).

In Levine, Kumar, *et al.*, 2020, the authors suggest that using the Maximum Mean Discrepancy (MMD) measure can fit well in this case. We will not go in further detail, but it is important to say that there is not one size fits all solution regarding policy constraints, and making sure the right restrictions are applied can make a difference.

3.2.1 CQL

One algorithm that recently has shown great performance is Conservative Q-Learning. Proposed in Kumar, Zhou, *et al.*, 2020, it consists of learning a conservative Q-function which contains a penalty term, thus avoiding overestimation for out-of-distribution actions and reducing the impact of distributional shift.

Given a policy π, the classical Q-function consists of:

$$Q^{\pi}(s,a) = r(s_t,a_t) + \gamma * E_{s_{t+1}\sim p(s_{t+1}|s_t,a_t),a_t+1\sim\pi(a_t+1|s_t+1)}[Q(s_{t+1},a_{t+1})] \quad (3.3)$$

or in terms of the Bellman operator B^{π}:

$$\vec{Q}^{\pi} = \vec{B}^{\pi}\vec{Q}^{\pi} \quad (3.4)$$

The algorithm for Q-learning, based on dynamic programming, takes the action that maximizes over the next state s_{t+1}, leading to the equation:

$$Q^*(s, a) = r(s_t, a_t) + \gamma * E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}[\max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})] \tag{3.5}$$

It is common to use deep neural networks to parametrize the Q function, so we define Q_θ. For policy evaluation, as described above, we will estimate Q for a given policy π using a dataset D, generated by the behavior policy $\pi_\beta(a|s)$, adding a conservative penalty term. It should mostly push down on Q-values for out-of-distribution actions for which the Q-values are (potentially erroneously) high.

The equation has a minimization over the expected Q-values under state-action distribution $\mu(a|s)$ and a maximization over the data distribution $\pi_\beta(a|s)$. Given the Q-function is queried over unseen actions, $\mu(a|s)$ is restricted to the state marginal of the dataset such that $\mu(s|a) = d^{\pi_\beta}(s)\mu(s|a)$.

Furthermore, it has a penalty - regularizer - term defined as $R(\mu)$, which for CQL(H) - a CQL variant discussed in Kumar, Zhou, *et al.*, 2020 -, is $R(\mu) = -D_{KL}(\mu, \rho)$ (the KL divergence against distribution ρ). These variants can be derived by changing R, and they lead to different objectives for training the Q-function. Finally, the equation for CQL(R) is:

$$\begin{aligned} min_Q max_\mu \alpha(E_{s\sim D, a\sim\mu(a|s)}[Q(s, a)] - E_{s\sim D, a\sim\hat{\pi}_\beta(a|s)}[Q(s, a)]) \\ + \frac{1}{2} E_{s,a,s'\sim D}[(Q(s, a) - \hat{B}^\pi_k \hat{Q}^k(s, a))^2] + R(\mu) \quad \text{for iteration } k \end{aligned} \tag{3.6}$$

Given this equation, we are now able to introduce it in the learning procedure. For CQL(H), given the definition of $R(\mu)$ as the KL divergence, we get:

$$\mu(a|s) \propto \rho(a|s) \exp(Q(s, a)) \tag{3.7}$$

Assuming $\rho = Unif(a)$, the first term of equation 3.6 is the same as a softmax of the Q-values at any state s. The authors also discuss having other regularizers and ρ as the previous policy $\hat{\pi}^{k-1}$, but claim CQL(H) is more stable with high dimension action spaces.

Then, we can define a minimization function, which will be used in the learning process further, as:

$$\begin{aligned} \min_Q \alpha E_{s\sim D}[\log \sum_a \exp Q(s, a) - E_{a\sim\hat{\pi}_\beta(a|s)}[Q(s, a)]] + \\ \frac{1}{2} E_{s,a,s'\sim D}[(Q(s, a) - \hat{B}^\pi_k \hat{Q}^k(s, a))^2] \end{aligned} \tag{3.8}$$

With the formulation introduced above, CQL can be used as a Q-learning or Actor

Critic algorithm, as it relies on the minimization problem on 3.8. By modifying the objective Q-function on a Actor Critic or Q-learning setting, with some small changes, it is already good to go, which is a good advantage.

We discussed previously about the types of constraints used in Offline RL, and CQL fits in the case of directly regularizing the value or Q function, introducing this new penalty term on the Actor Critic update.

Another important aspect is that it doesn't require modeling the behavior policy and representing the policy explicitly, because it modifies the value function introducing a penalty term that doesn't need it.

And as the authors discuss in the paper, it is possible to apply different types of penalties, such as CQL(ρ) - uses KL-divergence regularizer - and CQL(var.) - mentioned as variance regularized CQL - (Kumar, Zhou, *et al.*, 2020).

The authors compare CQL performance with BEAR (Kumar, Fu, *et al.*, 2019), BRAC (Wu *et al.*, 2019), SAC (Haarnoja *et al.*, 2018) and behavior cloning (BC), using different domains and datasets - where some use were generated by a random policy, expert examples and a mix of them.

In Gym (Brockman *et al.*, 2016) environments, they report 2x-3x performance over the mentioned algorithms. On some D4RL benchmark environments (Fu *et al.*, 2021), CQL and its variants (CQL(H)) report 2x-9x up performance over prior offline methods, showing significant results even on more difficult tasks such as Adroit tasks.

The CQL algorithm pseudocode:

Algorithm 5 Conservative Q-Learning pseudocode (Actor Critic variant)

Dataset $D = \{(s, a, s', r)_j\}$
Actor π_ϕ, Critic Q_θ
for iteration k = 1, 2, ... **do**
  $\theta_k = \theta_{k-1} - v_Q \nabla_\theta CQL(R)(\theta)$ ▷ Actor update
  $\phi_k = \phi k - 1 - v_\pi E_{s\sim D, a \sum \pi_\phi(.|s)}[Q_\theta(s, a) - \log \pi_\phi(a|s)]$ ▷ Critic update
end for

3.2.2 AWAC

AWAC (Advantage Weighted Actor Critic), proposed by Nair *et al.*, 2021, is an algorithm that approaches the RL learning problem by using both offline and online learning. It leverages large amounts of offline data, enabling using fixed datasets, and also enables online fine-tuning, incorporating the online training. Its main objective is to pre-train a model with a dataset D and use online interactions - few as possible - to refine and optimize the learned policy π.

The algorithm is build as an Actor Critic algorithm, fitting in the model-free off-policy methods. As describe in Actor Critic, they consist of an Actor, which acts (as the name suggests) given state s, and a Critic, responsible for quantifying, commonly with $Q(s, a)$,

the action chosen. It maximizes returns by alternating in 2 phases, policy evaluation and policy improvement.

As demonstrated by the authors, fine-tuning is performed off-policy, leading to a more data efficient method, when comparing with other that opt for on-policy fine-tuning. For on-policy fine-tuning, data reuse is less efficient on the online stage. Actor Critic algorithms, given their off-policy nature, fit this problem setting of not wasting previously collected data.

Nonetheless, standard Actor Critic methods have a hard time in the pure off-line setting. Given $Q(s', a')$, with $a' \sim \pi$, is used to update $Q(s, a)$, when a' is outside the data distribution, the target $Q(s', a')$ will be inaccurate, which leads to error to accumulation.

Prior offline RL algorithms that incorporate the Actor Critic architecture require estimating the behavior policy π_β and applying explicit constraints on the policy improvement step. The problem is that an accurate estimation of π_β is difficult.

The constraint is implemented on the policy improvement step. Given a dataset D, π_θ is the Actor Critic update, $\pi_\beta(a|s)$ the data generation distribution (offline and online data) and a divergence measure $\boldsymbol{D}$:

$$arg max_\theta E_{s\sim D}[E_{\pi_\theta(a|s)}[Q_{\phi_k}(s, a)]] \text{ s.t. } \boldsymbol{D}(\pi_\theta, \pi_\beta) \leq \epsilon \tag{3.9}$$

π_β is not known and must be estimated from the data. Other offline algorithms fit a parametric model to sample π_β via maximum likelihood estimation, taking uniformly random samples from it, leading to $\hat{\pi}_\beta = \max_{\hat{\pi}_\beta} E_{s,a\sim\pi_\beta}[\log \hat{\pi}_\beta(a|s)]$.

An accurate estimation of $\hat{\pi}_\beta$ is precisely where it is difficult to use standard Actor Critic methods with offline pre-training and online fine-tuning.

Furthermore, pure offline RL methods that rely on explicit constraints struggle with online fine-tuning. They tend to be over conservative and don't gain much performance with additional online steps (Nair *et al.*, 2021). The authors mention that this challenge can be attributed to fitting an accurate behavior model in the online process.

When they become inaccurate, the optimization step, which is explicitly constrained, becomes over conservative. Thus, improvement is not so noticeable. As we will discuss later, AWAC employs an implicit constraint on the optimization step, and mitigating the burden generated by its necessity.

For AWAC, the policy evaluation step applies a off-policy temporal difference learning update to estimate $Q^\pi(s, a)$, and a unique policy improvement step. It learns a policy that maximizes the Critic value in the policy evaluation via temporal difference bootstrapping, also restricting the policy distribution over the data observed along the Actor update. As mentioned before, this restriction is done implicitly, by deriving the optimization problem.

Given advantage function $A^{\pi}(s,a) = Q^{\pi}(s,a) - Q^{\pi}(s,a')$ where $a' \sim \pi_{\theta}(.|s)$:

$$\pi_{k+1} = argmax_{\pi \in \Pi} E_{a \sim \pi(.|s)}[A^{\pi_k}(s,a)] \text{ s.t. } D_{KL}(\pi(.|s)||\pi_{\beta}(.|s)) \leq \epsilon \tag{3.10}$$

The authors mention the derivation follows other previous works (Peng *et al.*, 2019 and Peters *et al.*, 2010).

To obtain an analytic solution it enforces the Karush-Kuhn-Tucker (KKT) conditions for solving a non linear constrained optimization problem. The complete process can be found in paper's Appendix section. The closed form solution for the optimization problem is:

$$\pi^*(a|s) = \pi_{\theta}(a|s) \frac{1}{Z(s)} exp\left(\frac{1}{\lambda} A^{\pi_k}(s,a)\right) \tag{3.11}$$

where $Z(s) = \int_a \pi_{\beta}(a|s) \exp 1/\lambda A^{\pi_k}(s,a) da$.

This solution is then projected to the policy space, done by minimizing the KL divergence of π_{θ} from the optimal non-parametric solution π^*. The authors mention using the forward KL direction as a key advantage, allowing to optimize θ as a maximum likelihood with expectation over the replay buffer β, not sampling action from the policy π_{β}, which may be out of distribution.

The policy improvement step is defined as:

$$\theta_{k+1} = argmax_{\theta} E_{s,a \sim \beta}\left[\log \pi_{\theta}(a|s) \frac{1}{Z(s)} exp\left(\frac{1}{\lambda} A^{\pi_k}(s,a)\right)\right] \tag{3.12}$$

It is worth mentioning the authors ignore $Z(s)$ in their implementation, affirming it leads to worth behavior. Also, using Q^{π} to estimate the advantage A^{π_k} improves on efficiency when comparing to AWR (Peng *et al.*, 2019), another method that is similar with AWAC.

This improvement step doesn't require modeling previously observed data and avoids the need of learning a behavior model, which reduces pure offline RL methods to learn from offline and online fine-tuning. Thus, AWAC performs better, as it is less conservative and more efficient on online fine-tuning.

The policy evaluation step, performed by the Critic, follows the traditional Bellman equation:

$$B^{\pi} Q(s,a) = r(s,a) + \gamma E_{p(s'|s,a)}[E_{\pi(a'|s')}[Q^{\pi}(s',a')]] \tag{3.13}$$

Which is minimized with respect to the Critic neural network parameters ϕ_k:

$$\phi_k = arg \min_{\phi} E_D[(Q_{\phi}(s,a) - (r(s,a) + \gamma E_{s',a'}[Q_{\phi_{k-1}}(s',a')]))^2] \tag{3.14}$$

With this formulation, AWAC is able to be data efficient, using an off-policy Critic, and avoids the bootstrap error accumulation, using a constrained Actor which doesn't need behavior policy estimation. The main point is that it enables learning faster than other methods, as it can use both online and offline data.

The AWAC algorithm pseudocode:

Algorithm 6 AWAC pseudocode

Dataset $D = \{(s, a, s', r)_j\}$
Replay buffer $\beta = D$
Define $num\_offline\_steps$
Actor π_θ, Critic Q_ϕ
for iteration k = 1, 2, ... **do**
  Sample batch $(s, a, s', r) \sim \beta$
  $\phi_k = \arg\min_\phi E_D[(Q_\phi(s, a) - (r(s, a) + \gamma E_{s',a'}[Q_{\phi_{k-1}}(s, a)])^2]$ ▷ Critic update
  $\theta_k = \arg\max_\theta E_{s,a\sim\beta}\left[\log \pi_\theta(a|s) \frac{1}{Z(s)} exp\left(\frac{1}{\lambda} A^{\pi_{k-1}}(s, a)\right)\right]$ ▷ Actor update
  if $i > num\_offline\_steps$ **then**
    $\tau_1, ..., \tau_k \sim p_{\pi_\theta}(\tau)$ ▷ Collect trajectories
    $\beta = \beta \cup \tau_1, ..., \tau_k$ ▷ Add trajectories to β
  end if
end for

3.3 Model-based approach

Model-based Reinforcement Learning is a term which relates to methods that explicitly learn the transition or dynamics function $T_\psi(s_{t+1}|s_t, a_t)$, parametrized by ψ. This learned model allows for the agent to gain information on its environment, and can be used for planning or training a policy.

The transition model may also be defined from its relationship with dataset D, $\hat{T}$ being estimated from transitions in D. As a consequence, for the model-based scenario, we have a MDP $\hat{M} = (S, A, \hat{T}, r, \mu_0, \gamma)$, similar to the definition in Preliminaries, but adding $\hat{T}$.

The need of modeling the dynamics model with a function approximator, in general, adds an overhead of time to model based algorithms, as it needs more compute power. Model-free algorithms, even when using an Actor Critic structure which relies commonly on 2 deep neural networks, don't rely on approximating the dynamics model. The algorithms we will discuss have another aspect that the implementation referred by the authors use an ensemble strategy, where multiple models are constructed with a different set of initialized weights and their results are averaged to obtain a more reliable result.

These multiple models also add an overhead, which again, need to be considered when thinking about model-based methods in the context of deep neural networks. This doesn't mean model-free methods are always and inevitably faster. Given they vary architectures,

how they estimate state or state-action values, the theoretical guarantees they have, they may be compute intensive as well.

Regarding Offline Model Based RL, a different set of challenges emerge. Although some of them are common to the model-free ones, like distributional shift, they can suffer from model exploitation and have other peculiarities. We will discuss them and also bring an algorithm that exemplifies how researches are approaching these challenges.

When dealing with a model, which is queried for a given state $s \sim d^{\pi}(s)$, where π is the resultant policy generated by planning under the model, distributional shift may arise. Given $d^{\pi}(s) \neq d^{\pi_\beta}(s)$, being $\pi_\beta(s)$ the behavior policy generated by dataset D, in other words, the difference between state visitation distribution for the learned and the behavior policy, the model may fall in distributional shift problem (Yu *et al.*, 2020).

Given we optimize our policy using the model, we may have erroneous values for out of distribution states and actions. Since the dataset may not have the entire state-action space, the model has an inherent inaccuracy (Kidambi *et al.*, 2021). Thus, planning using this model without any restrictions leads to what's called model exploitation. We may be too optimistic and be penalized when dealing with the real MDP, leading to worse performance than expected.

Nonetheless, model-based RL methods suit the offline setting even without many modifications from their original online approach, as we will see further when talking about MOPO (Yu *et al.*, 2020) and MORel (Kidambi *et al.*, 2021). In some cases, they outperform online model-free methods and even some already modified for the offline setting.

Most offline model-based methods approach the problem with adding some kind of conservative measure to how the values are calculated. There are different ways, such as modifying the MDP under the learned model or even the reward function, as this is usually related to uncertainty estimation. By assessing how our estimation is truthfully when compared to the "real" scenario, we can penalize our measurement, enforcing a more conservative behavior.

This relationship with uncertainty estimation poses new challenges, which in some cases can be hard. Furthermore, model-based reinforcement learning have difficulties when modeling high dimensional MDPs and long horizon estimates.

Note that one may see some differences on notation between the next sections and the original papers. The notation has been standardized across this work, so some modifications from the originals was required.

3.3.1 MOPO

To bring up the set of challenges and discuss offline model-based methods, lets discuss MOPO (Yu *et al.*, 2020) (Model-based Offline Policy Optimization). The main ingredient of this algorithm is to use an existing model-based method called MBPO (Janner *et al.*, 2019), incorporating conservative behavior based on an estimate of the model error.

This is accomplished by applying rewards artificially penalized, thus modifying the

underlying MDP reward function, using an uncertainty measure of the model.

Uncertainty estimation really is a main point for MOPO and it is what drives the algorithm and leverage its performance. The measure is in terms of an error oracle - an admissible error estimator - $u(s, a)$ which estimates the accuracy of the model for (s, a). The oracle needs to satisfy, given a divergence measure $D(.,.)$;

$$D(T_\psi(s_{t+1}|s_t, a_t)||T(s_{t+1}|s_t, a_t)) \leq u(s, a) \tag{3.15}$$

Then, it is possible to apply a conservative behavior on the modified reward function, penalizing uncertainty:

$$\tilde{r}(s, a) := r(s, a) - \lambda u(s, a) \tag{3.16}$$

And finally, we model an uncertainty-penalized MDP $\tilde{M} = (S, A, \hat{T}, \tilde{r}, \mu_0, \gamma)$.

The goal is to be better than what's on the data, but at the cost of becoming inaccurate. This inaccuracy cannot be corrected, as online data is not available. More precisely, there is a trade-off between gaining performance by also learning what's outside the behavior distribution with the risk of overfitting for regions where we know little about.

The solution the authors propose is to bound the return using the penalized MDP we discussed above, measuring the uncertainty of the model. This provides a policy which penalizes states where its likely to be incorrect but also maximizing this return.

The authors provide a number of theoretical guarantees for performance, showing bounds between the model-based estimate of the policy's performance and the true performance on the real MDP. Nonetheless, the implementation has some differences from the theoretical formulation, as the experiment showed that some aspects matter more than others, and some guarantees were loosen.

The practical implementation is basically MBPO with some distinctions, regarding uncertainty quantification. The dynamics is modeled using a neural network that outputs Gaussian distribution over next state and reward, as the reward function is unknown. An ensemble of N dynamics models are learned, each trained independently:

$$\text{Dynamics model: } \hat{T}_{\theta,\phi}(s_{t+1}, r|s_t, a_t) = \mathcal{N}(\mu_\theta(s_t, a_t), \Sigma_\phi(s_t, a_t))$$
$$\text{Ensemble: } \{\hat{T}^i_{\theta,\phi} = \mathcal{N}(\mu^i_\theta, \Sigma^i_\phi\}$$

The error estimator - error oracle - is designed as the maximum standard deviation of the models in the ensemble. This estimator doesn't rely directly on the theoretical guarantees, as the authors discuss on the paper, but has achieved good performance in practice:

$$u(s, a) = \max_{i=1}^{N} ||\Sigma_\phi^i(s, a)||$$

And the uncertainty penalized reward function $\tilde{r}$ with a penalty coefficient λ is:

$$\tilde{r} = \hat{r} - \lambda \max_{i=1,\dots,N} ||\Sigma_\phi^i(s, a)|| \text{ where } \hat{r} \text{ is the mean of the predicted reward for } \hat{T}$$

The comparison made by the authors relate to BEAR (Kumar, Fu, *et al.*, 2019, BC (Behavior Cloning), MBPO (Janner *et al.*, 2019), SAC (Haarnoja *et al.*, 2018) and BRAC (Wu *et al.*, 2019) using D4RL benchmarks (Fu *et al.*, 2021), but they do not compare it to CQL or AWAC, discussed above at Model-free approach.

This is generally not true for both CQL and AWAC, as they are constructed already considering this question and have shown state of the art results for the Gym environments presented in MOPO paper.

It is interesting and also commented by the authors that model-based methods, even with small or no changes to their default online implementations, exceed vanilla model-free methods. They suggest that model-based algorithms are more resilient to overestimation and overfitting than off-policy model-free RL algorithms.

In the experiments performed by the authors, MOPO shows great results. They test the algorithm using mixed datasets - collected by random policy, a policy trained with SAC, mix of fully and partially trained policy (*medium-mixed)* - and demonstrate that MOPO exceeds on 5 of 12 environment and dataset combinations. They affirm the most notable results are when using mixed and medium-mixed datasets.

The pseudocode for MOPO:

Algorithm 7 MOPO pseudocode

Require: reward penalty coefficient λ, rollout horizon h, rollout batch size b.
Dataset $D = \{(s, a, s', r)_j\}$
Train on D an ensemble of N probabilistic dynamics $\{\hat{T}^i(s', r|s, a) = N(\mu^i(s, a), \sum^i(s, a))\}$ for $i = 1, ..., N$
Replay buffer $\beta = \emptyset$
Policy π
for epoch e = 1, 2, ... **do**
  for t = 1, 2, ..., b **do**
    Sample $s_t \sim D$ to initialize rollout
    for j = 1, 2, ... h **do**
      $a_j \sim \pi(s_j)$
      Randomly pick $\hat{T}$ from $\{\hat{T}^i\}_{i=1}^N$
      Sample $s_{j+1}, r_j \sim \hat{T}(s_j, a_j)$
      Compute $\tilde{r}_j = r_j - \lambda \max_{i=1}^N || \sum^i(s_j, a_j)||$
      $\beta = \beta \cup (s_j, a_j, \tilde{r}_j, s_{j+1})$
    end for
  end for
  Draw samples from $D \cup \beta$, update π with SAC
end for

3.3.2 MORel

MORel (Model-based Offline Reinforcement Learning), proposed by Kidambi *et al.*, 2021, is, as the name suggests, an model-based approach to Offline RL. It was developed concurrently with MOPO, showing some similarities to it, and also has state-of-the-art results in many benchmark tasks, as we will see.

The main ingredient is constructing a uncertainty penalized MDP, but differently from MOPO, it measures uncertainty by partitioning the state space into "known" and "unknown" regions. Them, it can penalize the agent, using negative rewards, for unknown regions. The construction of this pessimistic MDP, referred as P-MDP, is a key idea in MORel.

The ability and necessity on having a form of penalizing unknown states helps on learning and policy evaluation, because as we mentioned earlier, the learned model may be severely impacted by model exploitation.

For MORel, uncertainty is measured through a function called α-USAD (Unknown state-action detector). It tells how confident the model is on a given state-action pair and it uses the total variation distance between $\hat{T}$ and T: $D_{TV} = (\hat{T}(.|s, a), T(.|s, a))$, and its defined as:

$$U^{\alpha}(s,a) = \begin{cases} False & \text{if } D_{TV} = (\hat{T}(.|s,a), T(.|s,a)) \leq \alpha \text{ can be guaranteed} \\ True & \text{otherwise} \end{cases} \tag{3.17}$$

The authors recall that for MORel to be effective is needs to have data points sufficiently close from where it is querying data and quality of the representations. They also suggest that larger datasets may enable stronger results, something that can be related to supervised learning as well, where having more data available may in general be better for performance, specially when using neural networks as function approximators.

The P-MDP is then constructed based on the learned dynamics model and this uncertainty measure USAD. The approximate dynamics model can be learned in different ways, using maximum likelihood estimation and other standard techniques.

Given USAD penalization, unknown states where the dynamics model, learned through a standard model-based procedure, are penalized thus limiting both distribution shift and model exploitation impacts.

The modified MDP $\hat{M}$ is then defined by the authors as $\hat{M} = (S \cup HALT, A, r_p, \hat{T}, \hat{\mu}_0, \gamma)$. S represents states, A represents actions, $HALT$ an absorbing state, $\hat{\mu}_0$ the initial state distribution fro D, γ is the discount factor. The transition $\hat{T}$ and reward function r_p as:

$$\hat{T}(s'|s,a) = \begin{cases} \delta(s' = HALT) & \text{if } U^{\alpha}(s,a) = TRUE \text{or } s = HALT \\ \hat{T}(s'|s,a) & \text{otherwise} \end{cases} \tag{3.18}$$

$$r_p(s,a) = \begin{cases} -\kappa & \text{if } s = HALT \\ r(s,a) & \text{otherwise} \end{cases} \tag{3.19}$$

Here, $\delta(s' = HALT)$ is the Dirac delta function, forcing the transition to the absorbing state in the case of the state-action pair being possibly unknown or when s already is the absorbing state.

The theoretical results which enable the construction of MORel reside on bounding the learned policy behavior. The authors show that the sub-optimality of the learned policy can be bounded, and also that is able to improve over the behavior policy as well - essential for an algorithm that needs to generalize and perform outside dataset D.

One point also worth mentioning is that having a broad dataset with more examples is important. The model can be more accurate and reliable when it has more information on the environment, because it can't interact online with it.

The implementation of MORel uses Gaussian dynamics models $\mathcal{N}(\mu_\theta(s,a), \Sigma_\phi(s_t, a_t))$, same as MOPO. USAD is used in uncertainty quantification along with ensembles of the Gaussian models, where multiple models are learned, each with different initialized weights,

and then measuring a disagreement measure, defined as $d(s, a) = \max_{i,j} ||\mu_{\theta_i}(s, a) - \mu_{\theta_j}(s, a)||_2$. USAD is finally defined as:

$$U_{practical}(s, a) = \begin{cases} False & \text{if } d(s, a) \leq \epsilon \\ True & \text{otherwise} \end{cases} \tag{3.20}$$

ϵ is a hyperparameter

The experiments provided by the authors were based on benchmark tasks from MuJoCo (Todorov *et al.*, 2012) and Gym (OpenAI *et al.*, 2019). They use 5 types of datasets - collected by different policies - each with 1 million of timesteps, following the same procedure of collection used in Wu *et al.*, 2019.

It exceeds BEAR (Kumar, Fu, *et al.*, 2019), BCQ (Fujimoto *et al.*, 2019) and BRAC (Wu *et al.*, 2019) on 12 out of 20 environments, mostly notably on MuJoCo walking tasks - where agents need to move efficiently through the environment.

Another interesting comparison the authors show is regarding D4RL benchmarks (Fu *et al.*, 2021). It had the best score on 5 of 12 domains, and the next best algorithm was CQL, with 3 of 12. MOPO resulted in 2 of 12, along with a variation of BRAC. These shows that they are very competitive and can achieve good results. More details on the scores, number of iterations and other specifics can be found on the original paper (Kidambi *et al.*, 2021).

Algorithm 8 MORel pseudocode

Require: dataset D
Train on D an ensemble of N probabilistic dynamics $\{\hat{T}^i(s', r|s, a) = N(\mu^i(s, a), \sum^i(s, a))\}$ for $i = 1, ..., N$
Construct uncertainty measure α-USAD U^α with D
Construct modified MDP $\hat{M}$
Construct policy π with a planner using $\hat{M}$ and π_β

Chapter 4

Conclusion

Offline Reinforcement Learning offers a different view on Reinforcement Learning. By using datasets with lots of data, as Computer Vision, Natural Language Processing areas did in the recent years, it opens Reinforcement Learning to new possibilities and capabilities, such as having robust models which can be further refined on special tasks, dealing with sensitive fields where real interaction is not allowed or too expensive.

Although not new as a research area, in the last couple of years, powerful algorithms have emerged and expanded the range of domains and environments were Offline RL can be applied. Also, they provide new ideas on how to approach the challenges inherent of this area, the most noticeable being distributional shift, as we discussed in the last chapter.

We saw 4 state-of-the-art algorithms that surpass over 2 different areas of Reinforcement Learning - model free and model based algorithms. AWAC and CQL leverage the use of a fixed dataset through changes in the formulation of the Actor Critic architecture - and recall CQL can also be used as a Q-learning algorithm - enabling efficient learning.

For CQL, it shows remarkable results when in classic MuJoCo environments (Todorov *et al.*, 2012) when comparing to SAC (Haarnoja *et al.*, 2018), BEAR Kumar, Fu, *et al.*, 2019), BRAC (Wu *et al.*, 2019) and its variants. It offers a practical implementation which leverages the Actor Critic architecture, by not explicitly using policy constraints, that with only small changes enables to use a fixed dataset.

In the case of AWAC, it goes a step further to also allow for online fine-tuning, which by the results presented by the authors, exceed the learning speed when comparing to methods such as BRAC (Wu *et al.*, 2019), SAC (Haarnoja *et al.*, 2018), ABM (Siegel *et al.*, 2020), and others in MuJoCo benchmark tasks.

As for MOPO and MORel, although similar, they emphasize the importance of uncertainty estimation under model based offline methods.

In MOPO, by also using an uncertainty penalized reward function, which introduces the desired conservative behavior when an agent has to decide what to do when it doesn't have much "information" - meaning a low uncertainty measure - on a given state (or state-action pair).

For MORel, it enforces the uncertainty estimation in a different manner, with a hard threshold to differentiate known and unknown states. Both of them tackle model exploitation, a particular issue for model-based methods when applied offline.

Another important aspect that leveraged recent Offline and Online Reinforcement Learning algorithms is the use of Deep Neural Networks as function approximators in Actor Critic architecture, calculating Policy Gradient and Value Functions. In the model-based approach, they also enable creating a model which has more capabilities of learning a bigger number of features in a given environment.

Of course, adding a non linear estimator with more complex data has its own set of challenges that need to be considered to make the algorithms reliable, but the benefit of having powerful neural networks has shown its benefits, as we also discussed in the previous chapters.

Offline Reinforcement Learning offers new perspectives, as new methods and algorithms are coming out, each one tackling a different aspect, challenge or domain. It is exciting to see the progress on performance for each of them, and also, as we saw with AWAC, using both offline and online scenarios to train an agent that tries to bring the best of both worlds. There is a lot of space left to make agents perform even better, as the current ones can struggle with balancing distributional shift and generalization outside the dataset.

The objective of this work was to give an broad view on this research area, showing state-of-the-art algorithms along with the reasoning behind them. The intention is to be a revision over the different approaches and how each of them handles the common set of problems for Offline RL. We also discussed the basics of Online Reinforcement Learning, going through Q-learning, Policy Gradients, and Actor Critic methods.

References

[Brockman *et al.* 2016] Greg Brockman *et al. OpenAI Gym.* 2016. arXiv: 1606.01540 [cs.LG] (cit. on pp. 6, 20).

[Chelba *et al.* 2014] Ciprian Chelba *et al. One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling.* 2014. arXiv: 1312.3005 [cs.CL] (cit. on pp. 2, 15).

[Covington *et al.* 2016] Paul Covington, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations". In: *Proceedings of the 10th ACM Conference on Recommender Systems.* RecSys '16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 191–198. isbn: 9781450340359. doi: 10.1145/2959100.2959190. url: https://doi.org/10.1145/2959100.2959190 (cit. on p. 1).

[Deng *et al.* 2009] Jia Deng *et al.* "Imagenet: a large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition.* 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848 (cit. on pp. 2, 15).

[Fu *et al.* 2021] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. *D4RL: Datasets for Deep Data-Driven Reinforcement Learning.* 2021. arXiv: 2004.07219 [cs.LG] (cit. on pp. 20, 26, 29).

[Fujimoto *et al.* 2019] Scott Fujimoto, David Meger, and Doina Precup. *Off-Policy Deep Reinforcement Learning without Exploration.* 2019. arXiv: 1812.02900 [cs.LG] (cit. on pp. 16, 29).

[Gottesman *et al.* 2018] Omer Gottesman *et al. Evaluating Reinforcement Learning Algorithms in Observational Health Settings.* 2018. arXiv: 1805.12298 [cs.LG] (cit. on pp. 2, 15).

[Haarnoja *et al.* 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.* 2018. arXiv: 1801.01290 [cs.LG] (cit. on pp. 7, 11, 20, 26, 31).

[Ioffe and Szegedy 2015] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG] (cit. on p. 13).

[Janner *et al.* 2019] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. *When to Trust Your Model: Model-Based Policy Optimization*. 2019. arXiv: 1906.08253 [cs.LG] (cit. on pp. 24, 26).

[Kidambi *et al.* 2021] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. *MOReL : Model-Based Offline Reinforcement Learning*. 2021. arXiv: 2005.05951 [cs.LG] (cit. on pp. 24, 27, 29).

[Kumar, Fu, *et al.* 2019] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. *Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction*. 2019. arXiv: 1906.00949 [cs.LG] (cit. on pp. 2, 7, 16–18, 20, 26, 29, 31).

[Kumar, Zhou, *et al.* 2020] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. *Conservative Q-Learning for Offline Reinforcement Learning*. 2020. arXiv: 2006.04779 [cs.LG] (cit. on pp. 16, 18–20).

[Lange S. 2012] Riedmiller M. Lange S. Gabel T. "Batch reinforcement learning". In: *Reinforcement Learning* 12 (2012). url: https://doi.org/10.1007/978-3-642-27645-3_2 (cit. on p. 2).

[Laroche *et al.* 2019] Romain Laroche, Paul Trichelair, and Rémi Tachet des Combes. *Safe Policy Improvement with Baseline Bootstrapping*. 2019. arXiv: 1712.06924 [cs.LG] (cit. on p. 18).

[LeCun *et al.* 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444 (cit. on p. 5).

[Levine, Finn, *et al.* 2016] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. *End-to-End Training of Deep Visuomotor Policies*. 2016. arXiv: 1504.00702 [cs.LG] (cit. on pp. 1, 15).

[Levine, Kumar, *et al.* 2020] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. 2020. arXiv: 2005.01643 [cs.LG] (cit. on pp. 2, 6, 16–18).

[Lillicrap *et al.* 2019] Timothy P. Lillicrap *et al. Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG] (cit. on pp. 7, 8, 11, 12).

[Mhaskar *et al.* 2017] Hrushikesh Narhar Mhaskar, Qianli Liao, and Tomaso A. Poggio. "When and why are deep networks better than shallow ones?" In: *AAAI*. 2017 (cit. on p. 5).

[Mnih *et al.* 2015] Volodymyr Mnih *et al.* "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236 (cit. on pp. 1, 12, 13).

[Nair *et al.* 2021] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. *AWAC: Accelerating Online Reinforcement Learning with Offline Datasets*. 2021. arXiv: 2006.09359 [cs.LG] (cit. on pp. 2, 3, 20, 21).

[OpenAI *et al.* 2019] OpenAI *et al. Learning Dexterous In-Hand Manipulation*. 2019. arXiv: 1808.00177 [cs.LG] (cit. on pp. 2, 29).

[Paszke *et al.* 2019] Adam Paszke *et al.* "Pytorch: an imperative style, high-performance deep learning library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach *et al.* Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on p. 16).

[Peng *et al.* 2019] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. *Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning*. 2019. arXiv: 1910.00177 [cs.LG] (cit. on p. 22).

[Peters *et al.* 2010] Jan Peters, Katharina Mülling, and Yasemin Altün. "Relative entropy policy search". In: *AAAI Conference on Artificial* (2010), pp. 1607–1612 (cit. on p. 22).

[Sallab *et al.* 2017] AhmadEL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. "Deep reinforcement learning framework for autonomous driving". In: *Electronic Imaging* 2017.19 (Jan. 2017), pp. 70–76. ISSN: 2470-1173. DOI: 10.2352/issn.2470-1173.2017.19.avm-023. URL: http://dx.doi.org/10.2352/ISSN.2470-1173.2017.19.AVM-023 (cit. on p. 2).

[Schulman *et al.* 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG] (cit. on p. 7).

[Seno and Imai 2021] Takuma Seno and Michita Imai. *d3rlpy: An Offline Deep Reinforcement Learning Library*. 2021. arXiv: 2111.03788 [cs.LG] (cit. on p. 16).

[Siegel *et al.* 2020] Noah Y. Siegel *et al. Keep Doing What Worked: Behavioral Modelling Priors for Offline Reinforcement Learning*. 2020. arXiv: 2002.08396 [cs.LG] (cit. on p. 31).

[Silver *et al.* 2017] David Silver *et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI] (cit. on p. 5).

[R.S. Sutton and Barto 2018] R.S. Sutton and A.G. Barto. *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018. ISBN: 9780262352703. URL: https://books.google.com.br/books?id=uWV0DwAAQBAJ (cit. on pp. 1, 5–8).

[Richard Sutton *et al.* 2000] Richard Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. "Policy gradient methods for reinforcement learning with function approximation". In: *Adv. Neural Inf. Process. Syst* 12 (Feb. 2000) (cit. on p. 10).

[Richard S Sutton *et al.* 2000] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*. 2000, pp. 1057–1063 (cit. on p. 9).

[Richard S. Sutton 1990] Richard S. Sutton. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Machine Learning Proceedings 1990*. Ed. by Bruce Porter and Raymond Mooney. San Francisco (CA): Morgan Kaufmann, 1990, pp. 216–224. ISBN: 978-1-55860-141-3. DOI: https://doi.org/10.1016/B978-1-55860-141-3.50030-4. URL: https://www.sciencedirect.com/science/article/pii/B9781558601413500304 (cit. on p. 8).

[Todorov *et al.* 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. *MuJoCo: A physics engine for model-based control*. 2012. DOI: 10.1109/IROS.2012.6386109 (cit. on pp. 29, 31).

[Watkins and Dayan 1992] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292 (cit. on pp. 7, 8).

[Wu *et al.* 2019] Yifan Wu, George Tucker, and Ofir Nachum. *Behavior Regularized Offline Reinforcement Learning*. 2019. arXiv: 1911.11361 [cs.LG] (cit. on pp. 16, 17, 20, 26, 29, 31).

[Yu *et al.* 2020] Tianhe Yu *et al. MOPO: Model-based Offline Policy Optimization*. 2020. arXiv: 2005.13239 [cs.LG] (cit. on p. 24).