ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Learning & Adaptive Systems

Master Thesis

DDACVaR: Distributional Deterministic Actor Critic approach for CVaR optimization in Reinforcement Learning

Spring Term 2020

Supervised by:
Sebastian Curi

Professor:
Andreas Krause

Author:
Núria Armengol Urpí

Contents

Abstract

Risk-sensitive decision-making provides a promising approach to compute robust and safe policies, essential in safety-critical applications such as healthcare or autonomous navigation. To this end, we present DDACVaR (Distributional Deterministic Actor-Critic approach for CVaR optimization), a model-free off-policy algorithm to find risk-sensitive policies that maximize the conditional Value-at-Risk (CVaR) of the return distribution. Adapting the distributional perspective on reinforcement learning to the continuous control setting, the critic learns a parameterization of the full action-value distribution instead of the traditional expected value. In its turn, the actor is updated towards maximizing the CVaR of such distribution. More remarkably, we also present a fully offline variant, O-DDACVaR algorithm, which can learn risk-sensitive policies in a fully offline setting. We demonstrate the effectiveness of O-DDACVaR in the domain of simulated robotics. Only making use of an external dataset, O-DDACVaR can learn risk-sensitive policies for several robotic environments that empirically reduce the probability of catastrophic events. This property holds tremendous promise for making possible to effectively allowing to turn any large enough dataset into a risk-sensitive policy without the need of exposing the agent to the risky environment during the training process. Additionally, we consider the algorithm can serve as a base for future algorithms that not only scale to real-world problems, but will also lead to solutions that generalize substantially better. We believe O-DDACVaR looks promising for advancing in the safe and generalizable data-driven RL area.

Chapter 1

Introduction

In sequential decision-making problems an agent interacts with an environment by selecting actions according to a policy and in turn, it observes the state transitions of the system and it receives a reward.
During this interaction different sources of irreducible randomness are introduced in the system due to modelling errors or inherent stochasticity of the environment. The most common optimization criterion for the decision maker to assess how good actions were, is the expectation of the cumulative reward with respect to the randomness in the system, which leads to the so called *risk-neutral* behavior. However, this criterion is not sensitive to the possible risks when the future return is stochastic (Tang et al., 2020).

The notion of risk is related to the fact that even an optimal policy with respect to the expected return may perform poorly in some cases. Since maximizing the expected return does not necessarily imply the avoidance of rare occurrences of large negative outcomes, we need other criteria to evaluate risk. (García and Fernández, 2015). Robust handling of uncertainties and risks is a must before we can leverage the power of decision-making agents in real world safety-critical applications (Shalev-Shwartz et al., 2017). Risk-sensitive decision-making, focused on learning robust and safe policies, provides a promising approach for such a goal. However, finding computationally tractable and conceptually meaningful methodologies is non-trivial and still a challenge.

In this thesis, we will work on the reinforcement learning (RL) framework in which a model of the environment is unknown and the agent must discover which actions yield the *best* reward by trying them. In most of the cases, actions may affect not only the immediate reward but also the next state and consequently, all subsequent rewards. Our approach aims to maximize another risk metric instead of expectation of cumulative rewards, namely the Conditional Value-at-Risk (CVaR). CVaR has recently gained a lot of popularity in various fields such as engineering or finance due to its mathematical properties (Artzner et al., 1999), which makes its computation and optimization much easier than for other metrics (Rockafellar and Uryasev, 2000). For example, it has recently been identified by Majumdar and Pavone (2020) as a suitable metric for measuring risk in robotics.

1.1 Related Work

Risk sensitive RL, safe RL and the related robust MDPs have been extensively studied in literature (Bagnell et al., 2001; Morimoto and Doya, 2005; Pinto et al., 2017). A recent comprehensive survey on the safe RL topic (García and Fernández, 2015) segments safe RL in two main categories: the first transforms the optimization criterion to introduce the concept of risk, whereas the second, modifies the exploration process to avoid exploratory actions that can lead to undesirable situations.
Our approach belongs to the first category, which can be divided into 3 subcategories: worst-case criterion, constrained criterion and risk-sensitive criterion.
The worst-case or minimax criterion has been studied by Heger (1994), Coraluppi and Marcus (2000) and Coraluppi and Marcus (1999), in which the objective is to compute a control policy that maximizes the expected return with respect to the worst case scenario. In general, minimax criterion has been found to be too restrictive as it takes into account severe but extremely rare events which may never occur.
The constrained criterion aims to maximize the expected return while keeping other types of expected utilities higher than some given bounds (Altman, 1993). It may be seen as finding the best policy π in the space of considered safe policies. This space may be restricted by different constraints such as ensuring that the expectation of costs (Geibel, 2006) or the variance of return don't exceed a given threshold (Tamar et al., 2012). These constraint problems can be converted to equivalent unconstrained ones by using penalty methods or a Lagrangian approach.
Finally, risk-sensitive criterion uses other utility metrics to be maximized instead of expectation of cumulative rewards. Lot of research has been done using exponential utility functions (Howard and Matheson, 1972; Chung and Sobel, 1987), linear combination of return and variance (Sato et al., 2001), among others.

Conditional Value-at-Risk (CVaR) is another criterion that is gaining a lot of popularity in various fields such as engineering and finance. Non-formally, CVaR of a return distribution at confidence level α can be defined as the expected cumulative reward of outcomes worse than the α-tail of the distribution.

CVaR optimization aims to find the parameters θ that maximizes $\text{CVaR}_\alpha(Z)$, where the return distribution Z is parameterized by a controllable parameter θ such that: $Z = f(X; \theta)$. In the simplest scenarios, where X is not dependant on θ, CVaR optimization may be solved using various approaches such as in Rockafellar and Uryasev (2000).
However, in RL, the tunable parameter θ also affects the distribution of X and therefore, the standard existing approaches for CVaR optimization are not suitable. Additionally, most of the work with such a goal has been done in the context of MDPs when the model is known by using dynamic programming methods (Chow et al., 2015; Petrik and Subramanian, 2012) but not much research has been done for the RL framework.

Combining CVaR with RL, Morimura et al. (2010) and Morimura, Tetsuro and Sugiyama, Masashi and Kashima, Hisashi and Hachiya, Hirotaka and Tanaka (2010) focused on estimating the density of the returns to handle CVaR risk criteria. However, the resulting distributional-SARSA-with-CVaR algorithm they propose has proved effectiveness only in very simple and discrete MDPs.

Tamar et al. (2015) proposed a policy gradient (PG) algorithm for CVaR optimization by deriving a sampling based estimator for the gradient of CVaR. However, they only considered continuous loss distributions and they used empirical α-quantile to

estimate VaR_α which is known to be a biased estimator for small samples. Chow and Ghavamzadeh (2014) also proposed a PG algorithm for mean-CVaR optimization.
However these approaches have several disadvantages. First, they directly estimate the gradients for the CVaR from the received rewards, suffering from high variance especially when the trajectories are long. This high variance is even more exacerbated when using very low quantiles α for the CVaR since the averaging on the rewards is effectively only on αN samples. Second, they are both very sample inefficient since only the rewards below the quantile are used to compute the gradients. Third, they are both trajectory-based (not incremental), i.e they only update the parameters after observing one or more trajectory roll-outs.
Chow and Ghavamzadeh (2014) also proposed an incremental actor-critic approach which helped in reducing the variance of PG algorithms. However, they require state augmentation of the original MDP formulation to a state-space $\mathcal{X} \times \mathcal{S}$ where $\mathcal{S} \in \mathbb{R}$ is an additional continuous state that allows to reformulate the Lagrangian objective function as an MDP. Despite this augmentation, they still need to make use of several approximations to allow the algorithm be fully incremental and avoid sparse updates of the parameters.

It is also important to be noticed that all the aforementioned algorithms are on-policy approaches. This is first another source of sample inefficiency because they cannot exploit data from experts or other sources. Additionally they constraint the learning process to happen online while interacting with the environment, which in real-case scenarios, it is paradoxically risky.

1.2 Our approach

In this thesis we present DDACVaR (Distributional Deterministic Actor-Critic approach for CVaR optimization), a model-free, off-policy deterministic actor-critic algorithm using deep function approximators for CVaR optimization. Instead of empirically estimating the CVaR from the observed rewards, we learn the full value distribution and estimate the CVaR by sampling from the learnt distribution.
To approach that, we build upon recent research in distributional RL (Bellemare et al., 2017; Dabney et al., 2018a,b) to estimate the full *action-value distribution* (i.e. the distribution of the random return received by a RL agent) and we adapt it to the continuous setting using deep deterministic policy gradients.

The CVaR optimization algorithm that we introduce has 2 remarkable properties, namely being an off-policy algorithm and using deterministic policies.
Being off-policy means that the algorithm can learn the optimal policy from data collected from another policy. Hence, we can use a more exploratory *behavior* policy to interact with the environment and collect observations, and then use them to train the optimal *target* policy. Additionally to the fact of making exploration easier compared to on-policy algorithms, being off-policy allows the possibility to learn from data generated by a conventional non-learning controller or from a human expert, namely learning under, how is called in literature, *Batch RL* or *offline RL* setting. This application is really appealing when working with risky environments, when we do not want to expose the learning agent to its risks at the earlier stages of the learning process.
With this goal, we build upon DDACVaR, and propose a second algorithm named O-DDACVaR (Offline-DDACVaR) which by introducing an action generative model, allows us to implement the CVaR optimization algorithm in a fully offline setting.

The second property is the fact that we use a target policy which is deterministic. Using stochastic policies is not natural in many applications and it could increase the variance of the return Taleghan and Dietterich (2018). When again, working in an stochastic environment and with the goal of finding risk-aware policies, using a policy which is stochastic sounds quite counterintuitive. Additionally, from a practical viewpoint, using stochastic policies requires integrating over both state and action spaces to compute the policy gradient, whereas the deterministic case only needs to integrate over the state space. Hence, stochastic policy gradients may require much more samples, especially if the action space has many dimensions (Silver et al., 2014).

In a similar line than ours, using a distributional RL approach, we would like to mention some recent research. Barth-Maron et al. (2018) also adapt the distributional perspective on RL to the continuous setting using deterministic policy gradients. However, the estimated distributions are not used to minimize any risk-sensitive criteria but still focus on maximizing the expected return. While Dabney et al. (2018b) do optimize for risk-sensitive measures, their approach is only valid for discrete action spaces. Finally, Tang et al. (2020) also propose a CVaR optimization using deterministic policy gradients and a distributional critic. However, their approach differs from ours since they decide to approximate the return distribution up to its second order-statistics, i.e represent it using its mean and variance. They model the distribution as a Gaussian and in that way they can compute a closed-from calculation for the CVaR. Despite they show its top performance in a simulated self-driving environment we consider that using a Gaussian approximation for the return distribution can be a bit restrictive, because it is light-tailed and unimodal. One of the motivations for using CVaR to assess risk instead of other metrics is the fact that it accounts for the distribution of the tails. Using a return distribution which is unimodal may lead to omission of bumps in the left-tail of the distribution which wastes one of the main properties of the CVaR as a metric.

We show performance of both DDACVaR and O-DDACVaR algorithms. The first in a setting in which the environment simulator is given and the agent can interact with it during training to collect new data. The second, and most importantly, in a setting in which the agent is trained completely offline by using an external dataset and when no further interaction with the environment is allowed.

We first test performance in a toy car example and then we upgrade towards more complex environments from D4RL (Fu et al., 2020), a recent suite of tasks and datasets for benchmarking progress in offline RL. Specially we focus on robotic tasks developed in the Mujoco Physics simulator (Todorov et al., 2012), but we are confident our approach could be extended to other areas, such as for medical applications or autonomous navigation. We show our algorithm ability to minimize risk in terms of tail performance by introducing stochasticity in the original cost function of the environments and comparing the policies learnt by the risk-averse setting and by the risk-neutral one.

We hereby confirm DDACVaR and O-DDACVaR algorithms are able to learn risk-sensitive policies in complex high-dimensional, continuous action spaces.

Chapter 2

Theoretical Background

2.1 Problem description

Standard RL algorithms aim to find policies which maximize the expected cumulative reward. However, this approach neither takes into account variability of the reward around its mean neither sensitivity of the policy to modelling errors.
In this thesis, we change the objective function of the standard RL approach to one that optimizes another metric of the reward distribution which takes into account the risk of the actions taken by the agent. While several metrics have been designed to assess risk, we will focus on a particular risk metric called Conditional Value at Risk (CVaR).
In this thesis we present DDACVaR and O-DDACVaR, an RL algorithm to find risk-sensitive policies that maximize the CVaR of the return distribution.

2.1.1 Reinforcement Learning (RL)

RL is an approach to learn a mapping from states to actions so as to maximize a numerical reward signal. The learner or *agent* is not told which actions to take but instead must discover which actions yield the most reward by trying them. In most of the cases, actions may affect not only the immediate reward but also the next state and consequently, all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning (Sutton and Barto, 1998).

Markovian Decision Processes (MDPs)

We formalize the problem of RL by using the framework of Markov decision processes (MDPs) to define the interaction between a learning agent and its environment in terms of states, actions and rewards.

MDPs are discrete-time stochastic control processes which provide a mathematical framework for modeling sequential decision making in situations where outcomes are partly random and partly under the control of a decision maker and where

actions influence not only immediate rewards but also future ones.
An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$, where $\mathcal{S}, \mathcal{A}$ are the state and action spaces respectively, $R(x, a)$ is the reward distribution, $P(\cdot|x, a)$ is the transition probability distribution and $\gamma \in [0, 1]$ is the discount factor.

The state transitions of an MDP satisfy the Markov property in which the set of transition probabilities to next states depend only on the current state and action of the system but are conditionally independent of all previous states and actions. Hence, the state must provide information about all aspects of the past agent-environment interaction that make a difference for the future.

Solving an MDP involves determining a sequence of policies π (mapping states to actions) that maximize an objective function. A commonly considered class of policies in literature are the class of Markovian policies Π_M where at each time step t the policy π_t is a function that maps state x_t to the probability distribution over the action space $\mathcal{A}$. In the special case when the policies are time-homogeneous, i.e. $\pi_j = \pi \ \forall j \geq 0$ then the class of policies is known as stationary Markovian $\Pi_{M,S}$.
This set of policies, under which actions only depend on current state information and its state-action mapping is time-independent, makes the problem of solving for an optimal policy more tractable and common solution techniques involve dynamic programming algorithms (Bertsekas, 1976) such as Bellman iteration.
When the objective function is given by a risk-neutral expectation of cumulative reward, i.e.:

$$\min_{\pi \in \Pi_H} \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t) | x_0, a_t \sim \pi_t(\cdot|h_t)\big] \tag{2.1}$$

where Π_H represents the set of all history-dependant policies, the Bellman's principle of optimality (Bertsekas, 1976) shows that the optimal policy lies in the class of stationary Markovian policies $\Pi_{M,S}$

When dealing with other types of objective functions that aim towards more risk-sensitive policies, these nice properties don't normally hold and require extra mathematical formulations, such as MDP state augmentation (Chow et al., 2015).

2.2 Risk

When aiming to maximize the expected cumulative reward the shape of the reward distribution might be unimportant since highly different distributions still can have same expectation. However, in real world scenarios where catastrophic losses can occur, probability of certain outcomes, or risk, must be taken into account.

We can find 3 types of strategies with respect to risk, namely *risk neutral, risk averse* and *risk seeking.* We use the following example to show difference between the 3 strategies. Suppose a participant in a TV game is told to choose between two doors. One door hides 1000CHF and the other 0CHF. The TV host also allows the contestant to take 500CHF instead of choosing a door. The two options (choosing between door 1 and door 2, or taking 500CHF) have the same expected value of 500CHF. But it can clearly be seen that the risk among two options is different. Since the expected value is the same, risk neutral contestant is indifferent between these choices. A risk-seeking contestant will maximize its utility from the uncertainty and hence choose a door, whereas the risk-averse contestant will accept the guaranteed 500CHF.

2.2.1 Conditional Value-at-Risk (CVaR)

Let Z be a bounded-mean random variable, i.e $\mathbb{E}[|Z|] < \infty$ on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, with cumulative distribution function $F_Z(z) = \mathbb{P}(Z \leq z)$. We interpret Z as the reward distribution and for convenience we assume that is a continuous random variable meaning that $F_Z(z)$ is everywhere continuous. The value-at-risk (VaR) at confidence level $\alpha \in (0, 1)$ is the α-quantile of Z, i.e.:

$$\mathrm{VaR}_\alpha(Z) = F_Z^{-1}(\alpha) = \inf\{z \mid F_Z(z) \geq \alpha\} \tag{2.2}$$

The conditional value-at-risk (CVaR) at confidence level $\alpha \in (0, 1)$ is defined as the expected reward of outcomes worse than the α-quantile (VaR_α):

$$\mathrm{CVaR}_\alpha(Z) = \mathbb{E}[Z|Z \leq \mathrm{VaR}_\alpha] = \frac{1}{\alpha}\int_0^\alpha F_Z^{-1}(\tau)d\tau = \frac{1}{\alpha}\int_0^\alpha \mathrm{VaR}_\tau(Z)d\tau \tag{2.3}$$

The last equality, known as the Acerbi's integral formula for CVaR (Proposition 3.2 in Acerbi and Tasche (2002)), interprets CVaR_α as the integral of all the quantiles below the corresponding quantile level α and will become useful later.

While both VaR and CVaR are risk measures, CVaR has superior mathematical properties (monotonicity, translation invariance, positive homogeneity and subadditivity; see Artzner et al. (1999)) which makes CVaR computation and optimization far easier compared to VaR (Rockafellar and Uryasev, 2000). In addition, CVaR takes into account the possibility of tail events where losses exceeds VaR whereas VaR is incapable of distinguishing situations beyond it.

Rockafellar and Uryasev (2000) also showed that CVaR is equivalent to the solution of the following optimization problem:

$$\mathrm{CVaR}_\alpha(Z) = \max_\nu\{\nu + \frac{1}{\alpha}\mathbb{E}_Z[[Z - \nu]^-]\} \tag{2.4}$$

where $(x)^- = \min(x, 0)$.

In the optimal point it holds that $\nu^* = \mathrm{VaR}_\alpha(Z)$.

A useful property of CVaR is its alternative dual representation Artzner et al. (1999):

$$\mathrm{CVaR}_\alpha(Z) = \min_{\xi \in U_{\mathrm{CVaR}}(\alpha, \mathbb{P})} \mathbb{E}_\xi[Z] \tag{2.5}$$

where $\mathbb{E}_\xi[Z]$ denotes the ξ-weighted expectation of Z and the risk envelope U_{CVaR} is given by:

$$U_{\mathrm{CVaR}}(\alpha, \mathbb{P}) = \Big\{\xi | \xi(w) \in \big[0, \frac{1}{\alpha}\big], \int_{w\in\Omega} \xi(w)\mathbb{P}(w)dw = 1\big]\Big\} \tag{2.6}$$

Thus, the CVaR of a random variable may be interpreted as the worst case expectation of Z, under a perturbed distribution $\xi\mathbb{P}$.

Our goal in this thesis is to find policies that maximize the CVaR of the return distribution i.e.:

$$\arg\max_\pi \mathrm{CVaR}_\alpha[Z(x, \pi(x))] \quad \forall x \in \mathcal{X} \tag{2.7}$$

To this end we present DDACVaR and O-DDACVaR, two algorithms for CVaR optimization using a *distributional* variant of the deterministic policy gradient algorithm.

The CVaR dual formulation and the Acerbi's integral formula 2.3 presented above, will be useful to derive our approach to compute the CVaR from parameterized inverse cumulative distributions.

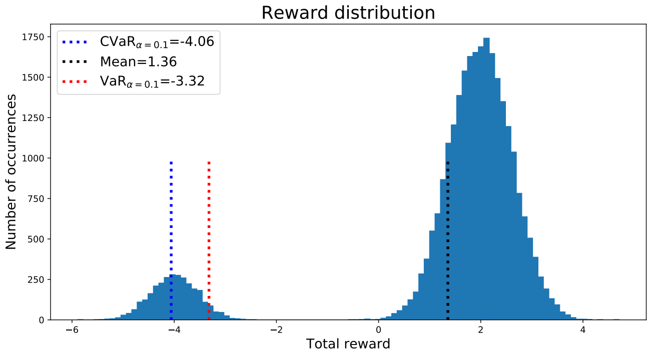

Figure 2.1: Histogram showing VaR, mean and CVaR of a sampled probability distribution. The image shows that the mean doesn't take into account low probability events and the flow of the VaR as a risk metric, since we could move the left-most mode to minus infinity and the VaR will remain the same, whereas the CVaR would change with the shift.

Chapter 3

The DDACVaR

We introduce the DDACVaR algorithm (Distributional Deterministic Actor-Critic algorithm for CVaR optimization), an off-policy, model-free algorithm for CVaR optimization using deep function approximators that can learn policies in high-dimensional, continuous action spaces.
The word *distributional* emphasizes that our approach takes inspiration from the recent advances in distributional RL (Bellemare et al., 2017; Dabney et al., 2018a,b) to estimate the full *action-value distribution*.
Additionally, we build upon DDPG (Lillicrap et al., 2016), a popular off-policy actor-critic using deep RL method for continuous control.

Specifically, our critic uses the distributional variant of RL and it is trained to learn a parameterization of the action-value distribution under current policy. Our actor maximizes the CVaR of this distribution by sampling from the parameterized value distribution to compute the CVaR and being updated using a variant of the deterministic policy gradient algorithm.

In 3.1 we briefly describe the standard DPG algorithm and its variants, then in 3.2 we introduce the DDACVaR algorithm and how we use it to obtain risk-sensitive policies, in 3.3 we focus on implementation details of the algorithm and we provide a pseudocode 1 and in 3.4 and 3.5 we present other alternatives for computing CVaR and other possible risk metrics that could be optimized without varying the algorithm. Extensive and more theoretical information on the distributional RL approach is addressed in appendix A.

3.1 Deterministic Policy Gradients

3.1.1 Preliminaries

The goal in standard RL is to learn a policy π^* which maximizes the expected return or (discounted) cumulative reward sampled from reward distribution $R(x, a)$ collected by the agent when acting in an environment with transition probability distribution $P(\cdot|x_t, a_t)$ starting from any initial state x. The action-value function for policy π, $Q^\pi(x, a)$, is used in many RL algorithms and describes the expected return after taking an action a in state x, and thereafter following policy π:

$$Q^{\pi}(x_t, a_t) = \mathbb{E}_{r_{i\geq t}\sim R, x_{i>t}\sim P, a_{i>t}\sim\pi}\Big[\sum_{i=t}^{T}\gamma^{(i-t)}r(x_i, a_i)\Big] \tag{3.1}$$

The Bellman's equation describes this value Q using the recursive relationship between the action-value of a state and the action-values of its successor states:

$$Q^{\pi}(x_t, a_t) = \mathbb{E}_{r_t\sim R, x_{t+1}\sim P}\Big[r(x_t, a_t)] + \gamma\mathbb{E}_{a_{t+1}\sim\pi}\big[Q^{\pi}(x_{t+1}, a_{t+1})\big]\Big] \tag{3.2}$$

The DPG algorithm (Silver et al., 2014) is characterized for using deterministic policies $a_t = \mu_\theta(x_t)$.
In general, behaving according to a deterministic policy does not ensure adequate exploration and may lead to suboptimal solutions. However, if the policy is deterministic we can remove the inner expectation with respect to a_{t+1} in 3.2 and then crucially, the expected cumulative reward in the next-state depends only on the environment and not on the policy distribution used to create the samples.

$$Q^{\pi}(x_t, a_t) = \mathbb{E}_{r_t\sim R, x_{t+1}\sim P}\Big[r(x_t, a_t) + \gamma Q^{\pi}(x_{t+1}, \pi(x_{t+1}))\Big] \tag{3.3}$$

This means that it is possible to learn the value function Q^{π} off-policy, i.e. using environment interactions which are generated by acting under a different stochastic behavior policy β (where $\beta \neq \pi$) which ensures enough exploration. An advantage of off-policy algorithms is that we can treat the problem of exploration independently from the learning algorithm.

To learn the optimal policy, Q-learning Watkins and Dayan (1992), a commonly used off-policy algorithm, first learns the optimal value function Q^* by iteratively applying the Bellman optimality operator to the current Q estimate:

$$Q(x_t, a_t) \leftarrow \mathbb{E}_{r_t\sim R, x_{t+1}\sim P}\Big[r(x_t, a_t) + \gamma\max_{a_{t+1}}Q(x_{t+1}, a_{t+1})\Big] \tag{3.4}$$

which is a contraction mapping proved to converge exponentially to Q^*, and then derives the optimal policy π^* from it via the greedy policy $a^* = \operatorname{argmax}_a Q^*(x, a)$.

When dealing with continuous actions, it is not possible to apply Q-learning straightforward because finding the greedy policy requires an optimization of a at every timestep, which is too slow to be practical with large action spaces. In this case, policy gradient methods are used in which a parameterized policy is learnt alongside the Q-function to be able to select actions without consulting the value function.

3.1.2 The Deterministic policy gradient

The deterministic policy gradient described by Silver et al. (2014) updates the parameters of the deterministic policy π_θ via gradient ascent to maximize an objective function $J(\pi_\theta)$:

$$J(\pi_\theta) = \mathbb{E}_{x\sim\rho^{\pi}}[Z(x, \pi_\theta(x))] \tag{3.5}$$

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{x\sim\rho^{\pi}}\left[\nabla_\theta\pi_\theta(x)\nabla_a Q^{\pi}(x, a)|_{a=\pi_\theta(x)}\right] \tag{3.6}$$

where ρ^{π} is the discounted state distribution when acting under policy π.

3.1.3 Off-policy Deterministic actor-critic

When we both learn approximations of the policy and the value function, the method is called deterministic actor-critic. Actor-critic algorithms combine the basic ideas from policy gradients and approximate dynamic programming. The actor is the learned policy which is updated with respect to the current value estimate, or critic. Sutton and Barto (1998).

Actor-critic methods are closely related with policy iteration methods (Lagoudakis and Parr, 2004) which consists of two phases: policy evaluation and policy improvement.
Unlike Q-learning, which directly attempts to learn the optimal Q-function, policy evaluation phase computes the Q-function for the current policy π, Q^π, by solving for the fixed point such that $Q^\pi = \mathcal{T}^\pi Q^\pi$. In the off-policy setting, as discussed previously, the critic (parameterized by θ^Q), can estimate the action-value function $Q^\pi(x, a)$ *off-policy* from trajectories generated by a behavior policy β using the Bellman equation. It learns by minimizing the loss:

$$\begin{aligned}\mathcal{L}(\theta^Q) &= \mathbb{E}_{x_t \sim \rho^\beta, a_t \sim \beta}\Big[(Q(x_t, a_t|\theta^Q) - y_t)^2\Big] \\ y_t &= r(x_t, a_t) + \gamma Q(x_{t+1}, \pi(x_{t+1})|\theta^Q)\end{aligned} \tag{3.7}$$

The policy improvement phase is done by choosing the action that greedily maximizes the Q-value at each state. With this aim, the actor (parameterized by θ^π) updates its parameters via gradient ascent by using the off-policy deterministic policy gradient Silver et al. (2014):

$$\begin{aligned}J_\beta(\pi|\theta^\pi) &= \int_{\mathcal{X}} \rho^\beta(s) Q^\pi(x, \pi(x|\theta^\pi))dx \\ \nabla_{\theta^\pi} J_\beta(\pi|\theta^\pi) &\approx \mathbb{E}_{x\sim\rho^\beta}\left[\nabla_{\theta^\pi}\pi(x, |\theta^\pi)\nabla_a Q^\pi(x, a)|_{a=\pi(x|\theta^\pi)}\right]\end{aligned} \tag{3.8}$$

where ρ^β is the discounted state distribution when acting under behavior policy β. By propagating the gradient through both policy and Q, the actor learns an approximation to the maximum of the value function under policy π averaged over the state distribution of the behavior policy β.
A term that depends on $\nabla_{\theta^\pi} Q^\pi(x, a)$ has been dropped in 3.8, following a justification given by Degris et al. (2012) that argues that this is a good approximation since it can preserve the set of local optima to which gradient ascent converges.

3.2 The DDACVaR

We present the DDACVaR algorithm, which is one of the main contributions of this thesis. The algorithm is based on the original off-policy deterministic actor critic explained in previous subsection 3.1.3, but introduces a distributional critic which estimates the whole value distribution instead of only its expected value. With this extra information, the actor can learn to maximize other metrics than the expected value, specifically the CVaR. We proceed to present the components of the algorithm.

3.2.1 Distributional Critic

We use a distributional variant of the standard critic function which maps from state-action pairs to distributions, inspired by the implicit quantile network (IQN) introduced in Dabney et al. (2018b).
IQN is a deterministic parametric function trained to reparameterize samples from a base distribution, e.g $\tau \in U([0,1])$, to the respective quantile values of a target distribution.
We define $Z(x,a)$ as the random variable representing the return, with cumulative distribution function $F(z) := P(Z \leq z)$ and we define $F_Z^{-1}(\tau) := Z(x,a;\tau)$ as its quantile function (or inverse cumulative distribution function) at $\tau \in [0,1]$. Thus, for $\tau \in U([0,1])$, the resulting state-action return distribution sample is $Z(x,a;\tau) \sim Z(x,a)$.
The IQN critic $Z(x,a;\tau)$ parameterized by θ^Z is hence a parametric function used to represent the quantile function at specific quantile levels.

As in Dabney et al. (2018b),we train the IQN critic using the sampled quantile regression loss (Koenker, 2005) on the pairwise temporal-difference (TD)-errors.
Quantile regression $\rho_\tau(u)$ is a method for approximating quantile functions of a distribution at specific points. Quantile regression loss for quantile $\tau \in [0,1]$ is an asymmetric convex lox function that penalizes underestimation errors (u<0) with weight $1-\tau$ and underestimation errors (u>0) with weight τ. See figure A.2 for a graphical representation.

For two samples $\tau, \tau' \sim U([0,1])$, current policy π (parameterized by θ^π) and replay buffer tuple (x_t, a_t, x_{t+1}, r_t) the sampled TD error is:

$$\delta^{\tau,\tau'} = r_t + \gamma Z(x_{t+1}, \pi(x_{t+1}); \tau') - Z(x_t, a_t; \tau) \tag{3.9}$$

Then, we compute the loss over the quantile samples:

$$\mathcal{L}_{QR}(x_t, a_t, r_t, x_{t+1}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho^\kappa_{\tau_i}(\delta^{\tau_i, \tau'_j}) \tag{3.10}$$

where N and N' are the number of iid samples $\tau_i, \tau'_j \sim U([0,1]$ used to estimate the loss and where ρ^κ_τ is the quantile Huber loss.
Quantile Huber loss acts as an asymmetric squared loss in an interval $[-\kappa, \kappa]$ around zero and reverts to a standard quantile loss outside this interval.
The Huber loss (Huber, 1964) is given by:

$$\mathcal{L}_\kappa(u) = \begin{cases} \frac{1}{2}u^2 & \text{if } |u| \leq \kappa \\ \kappa(|u| - \frac{1}{2}\kappa) & \text{otherwise} \end{cases} \tag{3.11}$$

Then the quantile Huber loss is the asymmetric variant of the Huber loss:

$$\rho^\kappa_\tau(u) = \Big|\tau - \mathbb{1}_{\{u<0\}}\Big| \mathcal{L}_\kappa(u), \forall u \in \mathbb{R} \tag{3.12}$$

The loss 3.10 gives unbiased sample gradients and hence we can minimize it with respect to θ^Z by stochastic gradient descent. Doing so, we find the parameters θ^Z to estimate the true quantile function.
For additional information and reasoning on the approach we address the reader to the appendix section A.1.3.

3.2.2 Actor

We update the policy via deterministic policy gradient ascent. We modify equation (3.8), to include the action-value distribution.

$$\nabla_{\theta^\pi} J_\beta(\pi|\theta^\pi) \approx \mathbb{E}_{x\sim\rho^\beta}\left[\nabla_{\theta^\pi}\pi(x,|\theta^\pi)\nabla_a Q^\pi(x,a)|_{a=\pi(x|\theta^\pi)}\right] \tag{3.13}$$

$$= \mathbb{E}_{x\sim\rho^\beta}\left[\nabla_{\theta^\pi}\pi(x,|\theta^\pi)\mathbb{E}[\nabla_a Z^\pi(x,a|\theta^Z)]|_{a=\pi(x|\theta^\pi)}\right] \tag{3.14}$$

The step from (3.13) to (3.14) comes by the fact that

$$Q^\pi(x,a) = \mathbb{E}[Z^\pi(x,a)] \tag{3.15}$$

With our goal of CVaR optimization in mind:

$$\arg\max_\pi \mathrm{CVaR}_\alpha[Z(x,\pi(x))] \quad \forall x \in \mathcal{X} \tag{3.16}$$

we can make use of the information provided by the Z distribution to optimize other objective functions rather than the expected value.
To approach this, we use as a performance objective the distorted expectation of Z(x,a) under the distortion risk measure $\phi : [0,1] \to [0,1]$, with identity corresponding to risk-neutrality, i.e.:

$$Q_\phi(x,a) = \mathbb{E}_{\tau\sim U([0,1])}[Z_{\phi(\tau)}(x,a)] \tag{3.17}$$

which is equivalent to the expected value of $F^{-1}_{Z(x,a)}$ weighted by ϕ, i.e.:

$$Q_\phi(x,a) = \int_0^1 F_Z^{-1}(\tau)d\phi(\tau) \tag{3.18}$$

When we use $\phi(\tau) = \alpha\tau$ as a mapping, 3.18 corresponds to the CVaR of Z(x,a) as already presented in 2.3, and as a reminder:

$$\mathrm{CVaR}_\alpha(Z) = \frac{1}{\alpha}\int_0^\alpha F_Z^{-1}(\tau)d\tau \tag{3.19}$$

Again, 3.19 is the Acerbi's integral formula for CVaR, which states that CVaR at confidence level α can be interpreted as the integral of all the quantiles below the corresponding α. We can hence approximate (3.19) via sampling, by taking K samples of $\tau \sim U[0,\alpha]$:

$$\mathrm{CVaR}_\alpha(Z) \approx \frac{1}{\alpha}\frac{1}{K}\sum_{i=1}^{K} Z(x,a;\tau_i) \qquad \tau_i \sim U[0,\alpha] \quad \forall i \in [1,K] \tag{3.20}$$

Therefore, we arrive to the formula for the DDACVaR actor parameters θ^π update:

$$J_\beta^{CVAR}(\pi) = \int_{\mathcal{X}} \rho^\beta(x)\mathrm{CVaR}_\alpha\Big(Z^\pi(x,\pi(x))\Big)dx$$

$$\nabla_{\theta^\pi} J_\beta(\pi) \approx \mathbb{E}_{x\sim\rho^\beta}\left[\nabla_{\theta^\pi}\pi(x)\nabla_a[\frac{1}{\alpha}\frac{1}{K}\sum_{i=1}^{K} Z(x,a;\tau_i)]|_{a=\pi(x)}\right] \tag{3.21}$$

where $\tau_i \in U([0,\alpha])\ \forall i \in [1,K]$

3.2.3 Summary

The algorithm can be summed up with:

1. Update distributional critic IQN via off-policy quantile-regression TD-learning:
$$\delta^{\tau,\tau'} = r + \gamma Z(x_{t+1}, \pi(x_{t+1}); \tau') - Z(x_t, a_t; \tau) \tag{3.22}$$

2. Update actor via deterministic risk-sensitive policy gradient ascent:
$$\nabla_{\theta^\pi} J_\beta(\pi) \approx \mathbb{E}_{x\sim\rho^\beta}\left[\nabla_{\theta^\pi}\pi(x)\nabla_a[\frac{1}{\alpha}\frac{1}{K}\sum_{i=1}^{K} Z(x, a; \tau_i)]|_{a=\pi(x)}\right] \tag{3.23}$$

Having the two main steps above, we again remark the capability of the algorithm to be implemented off-policy. We act in the environment using a more-exploratory behavior policy β and we learn a target policy π (where $\beta \neq \pi$). In contrast to stochastic off-policy actor-critic algorithms (Degris et al., 2012), we can avoid importance sampling both in the actor and the critic. This is thanks to the deterministic essence of the policies, which removes the integral over actions in the policy gradient for the actor and removes the expected value over actions in the Bellman equation for the critic.

3.3 Technical details of the algorithm

The algorithm uses neural networks as non-linear function approximators for both the actor and the critic. To make effective use of large neural networks, we use insights fom the DeepDPG algorithm Lillicrap et al. (2016) and in its turn from Deep Q Network (DQN) (Mnih et al., 2015). Before DQN, RL was believed to be unstable or diverge when nonlinear function approximators such as neural networks were used. The instability has several causes. First, correlation in the sequence of observations occurring when samples ar generated from exploring sequentially in an environment (hence i.i.d assumption is violated). Second, non-stationary data distribution since small updates in Q network may significantly change the policy and hereby, the data distribution. And third, correlation between the action-values and the target values during temporal difference backups since same network is being used. These 3 issues are solved by adding 2 innovations: 1) *experience replay buffer*: the network is trained by sampling batches of random past observations from a buffer, hereby removing correlations in the observation sequence and smoothing over changes in the data distribution. 2) use *target Q networks* that are only periodically updated, to update the Q network during temporal difference backups to reduce correlations with the target.

The agent perceives observations (x_t, a_t, r_t, x_{t+1}) when acting on the environment under behavior policy β and stores them in a fixed-sized (we used size of 10^6) buffer. When full, oldest samples are discarded. Since the algorithm is off-policy, the buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions. We address exploration by constructing an exploratory behavior policy β by adding noise η to the actor policy π. We below explicitly write $\pi(x_t|\theta^\pi)$ to remark that we use same neural network parameterized by θ^Z to compute the behavior actions.

$$\beta(x_t) = \pi(x_t|\theta^\pi) + \eta \tag{3.24}$$

where η is sampled from a noise process. In our case, as in Lillicrap et al. (2016),we used an Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930) with $\theta = 0.15$ and $\sigma = 0.3$ with exponential decay. Ornstein-Uhlenbeck process models the velocity of a Brownian particle with friction, which results in temporally correlated values centered around the mean (μ=0).

At every training step, we sample a minibatch of observations from the buffer. To estimate the target value for critic training, we use two target networks: *critic target network* $Z'(x, a; \tau)$ and *actor target network* $\pi'(x)$ (parameterized by $\theta^{Z'}$ and $\theta^{\pi'}$ respectively) which are initialized like their counterparts, but constrained to slowly track the learnt networks so that to stabilize learning. With this goal, weights of target networks θ' are updated "softly" via: $\theta' \leftarrow (1-\tau)\theta' + \tau\theta$ where $\tau \in [0, 1], \tau \lll 1$.

Additionally, we use two insights from TD3 Fujimoto et al. (2018). First, we update the policy and target networks less frequently than the critic network. As Fujimoto et al. (2018) recommends, we do one policy and target network updates for every two critic network updates.
Second, we use target policy smoothing to address a particular failure mode that can happen in DDPG (Lillicrap et al., 2016). To prevent policy to exploit actions for which the critic overestimated its value, target policy smoothing adds noise to target actions to smooth out Q over similar actions. Specifically, the target actions used for the TD backup are:

$$a_{t+1} = \pi'(x_{t+1}) + \text{clip}(\epsilon, -c, c) \quad \epsilon \sim \mathcal{N}(0, \sigma) \tag{3.25}$$

where we experimentally set $c = 0.5$ and $\sigma = 0.2$ and $\mathcal{N}(0, \sigma)$ is a Gaussian distribution with μ=0 and standard deviation σ.

To compute the quantile regression loss, we used N'=N=32 quantile levels to sample from target critic and critic networks. This parameter must be kept relatively big but further increasing it doesn't seem to improve performance. To compute the empirical CVaR from the estimated value distribution $Z(x, a; \tau)$, we use K=8 quantile levels.

We start training after a warm-up of T=25000 environment interactions. During this time, in order to highly enhance exploration we don't use the behavior policy but a random policy instead.

The actor and critic network parameters are updated using stochastic gradient ascent and descent, with learning rates of 10^{-4} and 10^{-3} for the actor and the critic, respectively, and adjusted online using Adam optimizer (Kingma and Ba, 2015). Algorithm pseudocode is provided below 1.

```
Algorithm 1: DDACVaR
Input: Replay buffer max size $\mathcal{B}$, quantile levels $N', N, K$, minibatch size $n$, $\alpha$
  confidence, $\eta$ exploration noise, $\{c, \epsilon, \sigma\}$ target policy smoothing parameters
Initialize: Critic network $Z$ and actor network $\Pi$ parameters and target
  networks $Z' \leftarrow Z$, $\Pi' \leftarrow \Pi$ parameters
while t=1:T do
    Collect data samples via environment interaction:
    if number batch data samples > $\frac{B}{4}$ then
        $a = \Pi(x) + \eta$
        $x', r, done = \text{act}(a)$
        buffer.add$(x, a, x', r, done)$
        training=True
    else
        $a = \text{random}$
        $x', r, done = \text{act}(a)$
        buffer.add$(x, a, x', r, done)$
        training = False
    end
    if training then
        Sample minibatch of $n$ transitions $(x_{\mathcal{B}}, a_{\mathcal{B}}, x'_{\mathcal{B}}, r_{\mathcal{B}})$ from the replay
          buffer $\mathcal{B}$
        Train critic:
        $\tau_i, \tau'_j \sim U([0,1]) \quad 1 \leq i, j \leq N, N'$
        Compute next action a':
        $a' = \Pi(x'_{\mathcal{B}}) + \text{clip}(\epsilon, -c, c) \quad \epsilon \sim \mathcal{N}(0, \sigma)$
        Compute distributional TD: $\delta_{ij} = r + \gamma Z'(x'_{\mathcal{B}}, a'; \tau'_j) - Z(x_{\mathcal{B}}, a_{\mathcal{B}}; \tau_i)\ \forall i, j$
        $\theta^Z \leftarrow \text{argmin}_{\theta_Z} \sum_{i=1}^{N} \mathbb{E}_{\tau'}[\rho_{\tau_i}(\delta_{ij})]$
        Train actor:
        $\tau_k \sim U([0, \alpha]) \quad 1 \leq k \leq K$
        Compute $\text{CVaR}_\alpha = \frac{1}{\alpha K} \sum_{k=1}^{K} Z(x_{\mathcal{B}}, a; \tau_k)$
        $\theta^\Pi \leftarrow \text{argmax}_{\theta_\Pi} CVaR_\alpha$
        Update target networks:
        $\theta^{Z'} \leftarrow \tau\theta^Z + (1-\tau)\theta^{Z'}$; $\theta^{\xi'} \leftarrow \tau\theta^\Pi + (1-\tau)\theta^{\Pi'}$
    end
end
```

3.4 Other approach to compute CVaR

When changing the sampling distribution for τ to $U([0, \alpha])$ to compute the CVaR of the distribution, we make use of both the dual formulation 2.5 and Acerbi's integral formulas for CVaR.3.19.

A second approach would be to use the Rockafellar and Uryasev (2000) optimization problem for CVaR presented in 2.4 and as a reminder:

$$\text{CVaR}_\alpha(Z) = \max_\nu \big\{\nu + \frac{1}{\alpha}\mathbb{E}_Z[[Z - \nu]^-]\big\} \tag{3.26}$$

which in the optimal point it holds that $\nu^* = \text{VaR}_\alpha(Z)$.

Given the fact that our critic network estimates the $\text{VaR}_\alpha(Z)$ of the return distribution, we can compute the CVaR using 3.26 by sampling uniformly from the whole quantile distribution, substracting the current estimated VaR for the confidence level α and truncating the result, that is to say, we can compute CVaR via sampling by:

$$\mathrm{CVaR}_{\alpha}(Z) \approx \nu + \frac{1}{\alpha}\frac{1}{K}\sum_{i=1}^{K}\left[Z(x,a;\tau_i|\theta^Z) - \nu\right]^{-} \tag{3.27}$$

where $\tau_i \sim U([0,1])$ and $\nu = Z(x,a;\alpha|\theta^Z)$.

A disadvantage of such approach is sample-inefficiency, since due to truncation, only approximately αK samples will be used to compute the policy gradient. This is a shared disadvantage with the algorithm presented in Chow and Ghavamzadeh (2014).

3.5 Other distortion risk measures

In this section we present other interesting distortion risk measures ϕ that can be used to find policies optimizing other metrics rather than the CVaR. As discussed, evaluating under different distortion risk measures is equivalent to changing the distribution used to sample the quantile levels τ. The fact of being able to switch the objective function just by changing the sampling distribution for τ gives to the algorithm a lot of versatility. Algorithms that directly derive policies for CVaR optimization (Chow and Ghavamzadeh, 2014; Tamar et al., 2015) don't show such property.

For the CVaR we have the distribution:

$$\mathrm{CVaR}(\alpha,\tau) = \alpha\tau \quad \tau \sim U([0,1]) \tag{3.28}$$

A distortion risk measure proposed by Wang (2000) can easily switch between risk-averse (for $\eta < 0$) and risk-sensitive (for $\eta > 0$) distortions:

$$\mathrm{Wang}(\eta,\tau) = \Phi(\Phi^{-1}(\tau) + \eta) \quad \tau \sim U([0,1]) \tag{3.29}$$

where Φ is the cumulative distribution of the standard Normal distribution.
While both Wang and CVaR heavily shift the distribution mass towards the tails of the distribution, CVaR entirely ignores all values corresponding to $\tau > \alpha$, whereas Wang gives to these non-zero, but vanishingly small probability (Dabney et al., 2018b).
Another, distortion risk measure would be a simple power formula for risk-averse (for $\eta < 0$) or risk-seeking(for $\eta > 0$) policies:

$$\mathrm{Pow}(\eta,\tau) = \begin{cases} \tau^{\frac{1}{1+|\eta|}} & \text{if } \eta \geq 0 \\ 1-(1-\tau)^{\frac{1}{1+|\eta|}} & \text{otherwise} \end{cases} \quad \tau \sim U([0,1]) \tag{3.30}$$

Chapter 4

The O-DDACVaR

4.1 Batch RL

We decide to test the capabilities of our algorithm in a *fully* off-policy setting, also called *batch RL setting* or *offline RL*. In this setting, the agent can only learn from a fixed dataset without further interaction with the environment.

Most of RL algorithms provide a fundamentally *online* learning paradigm which involves iteratively collecting experience by interacting with the environment and then using that experience to improve the policy. In many real-world scenarios, continuous interacting with the environment is impractical, either because data collection is expensive, as in robotics or healthcare, or dangerous, for example a physical robot may get its hardware damaged or damage surrounding objects. Additionally, even in some cases were online interaction is feasible, we might prefer to still use previously collected datasets - for example, if the domain is complex (such as a robot learning to cook several meals at home) and effective generalization (e.g cooking in different kitchens or using different ingredients for the meal) requires large datasets Levine et al. (2020).

The "off-policy" algorithm we presented in previous sections and similarly to most of the "off-policy" algorithms in literature, falls in the category of off-policy *"growing batch learning"*. In this case, agent's experience is appended to a data buffer $\mathcal{B}$ and each new policy π_k collects additional data, such that $\mathcal{B}$ is composed of samples from $\pi_0, \pi_1 ..., \pi_k$ and all of this data is used to train an update new policy π_{k+1}. Hence, despite being off-policy these methods require active online data collection.

In contrast, in offline RL the agent no longer has ability to interact with the environment and collect additional transitions. Instead, it can only employ a static dataset $\mathcal{B}$ collected by an external agent using policy π_β and must learn the best policy it can only using this dataset.

In numerous real-world applications, such as games or robotics, there is already plentiful amounts of previously collected interaction data which are a rich source of prior information. RL algorithms that can train agents using these prior datasets without further data collection will not only scale to real-world problems, but will also lead to solutions that generalize substantially better. Offline RL holds tremendous promise for making possible to turn large datasets into powerful decision-making engines, effectively allowing anyone with a large enough dataset to turn this dataset into a policy than can optimize a desired utility criterion. (Levine et al., 2020).

4.2 Issues with Batch RL

Most of recent off-policy algorithms such as Soft Actor-Critic (Haarnoja et al., 2018), DDPG (Lillicrap et al., 2016) and Rainbow (Hessel et al., 2017) could be in principle used as an offline RL algorithm, i.e. learn from data collected from an unknown behavioral policy π_β with state visitation frequency $\rho^\beta(s)$. However, in practice, they still require substantial amounts of “on-policy” data from the current behavioral policy in order to learn effectively and generally they fail to learn in the offline setting.

This is due to a fundamental problem of off-policy RL, called distributional shift, which in its turn induces what's called extrapolation error (Fujimoto et al., 2019) or bootstrapping error Kumar et al. (2019).
Distributional shift affects offline RL via dynamic programming algorithms, both at test time and training time. State distribution shift affects test-time performance, but no training, since neither the policy nor the Q-function is ever evaluated at any state that was not sampled from ρ_β.
However, training process is indeed affected by action distribution shift. This is because when doing the Bellman backup to update the Q-function estimate, the target values require evaluating the estimated $Q^\pi(x_{k+1}, a_{k+1})$ where a is chosen according to current policy π. Since targets are computed via bootstrapping, accuracy of Q-function regression depends on the estimate of the Q-value at these actions. When π differs substantially from π_β, these actions are generally unlikely or not contained in the dataset, i.e. out of the distribution of actions that Q-function was ever trained on. This can result in highly erroneous targets Q-values, and in its turn, in updated new Q estimates with pathological values that incur large absolute error from the optimal desired Q-value.
This issue is further exacerbated when π is explicitly optimized to maximize $Q^\pi(s, \pi(a))$. Then, the policy will learn to produce out-of-distribution (OOD) actions for which the learnt Q-function erroneously overestimates. This source of distribution shift is in fact one of the largest obstacles for practical application of dynamic programming methods to offline RL.

It is important to notice that for on-policy settings, extrapolation error is generally something positive since it leads to a beneficial exploration. In this case, if the value function is overestimating the value at a (state-action) pair, the current policy will lead the agent to that pair and will observe that in fact it is not as good and hence, the value estimate will be corrected afterwards. However, in the offline setting, the correction step on such over-optimistic Q-values cannot be done by the policy, due to the inability of collecting new data, and these errors accumulate over each iteration of training, resulting in arbitrarily poor final results or even divergence.

To address this OOD actions problems and effectively implement offline RL via dynamic programming, Fujimoto et al. (2018) opted for constraining the trained policy distribution to lie close to the dataset policy distribution, to avoid erroneous Q-values due to extrapolation error. Kumar et al. (2019) suggested a less restrictive solution to constrain the learnt policy to lie within the support of the dataset policy distribution.

For our algorithm, we will inspire ourselves in the approach presented in Fujimoto et al. (2019), which we introduce below with its further modifications.

4.3 Background on conditional variational autoencoders (VAE)

Fujimoto et al. (2019) presented a generative model G_w to generate actions with high similarity to the dataset. The generative model G_w is modelled via a conditional variational autoencoder Kingma and Welling (2014). Given state-action pairs (x_i^B, a_i^B) from the dataset, the conditional variational autoencoder generates action samples as a reasonable approximation to $\underset{a}{\text{argmax}} P_{\mathcal{B}}^G(a_i|x_i)$, where $P_{\mathcal{B}}^G(a_i|x_i)$ is the state-conditioned marginal likelihood given the (state-action) pairs in the batch $\mathcal{B}$. Specifically, given the state-condition action dataset $A|X = \{a_1|x_1, ..., a_N|x_N\}$, a conditional VAE aims to maximize the state-conditioned marginal log-likelihood $\log P_{\mathcal{B}}(A|X) = \sum_{i=1}^N \log P_{\mathcal{B}}(a_i|x_i)$.

It is assumed that the data is generated by some random process involving an unobserved continuous random variable $\mathbf{z}$. The process consists of two steps: (1) a value z_i is generated from some prior distribution p(z) and (2) an action a_i is generated from some conditional distribution $p(a|z,x)$. The state x is also included given the fact we are modelling the state-conditioned likelihood $P_{\mathcal{B}}^G(a|x)$. Given that the true probabilities are unknown, a recognition model $q(z|a,x;\phi)$ is introduced as an approximation to the intractable true posterior $p(z|a,x;\theta)$.
The recognition model $q(z|a,x;\phi)$ is called an *encoder*, since given a datapoint (x,a) it produces a *random latent vector* z. $p(a|z,x;\theta)$ is called a *decoder*, since given the random latent vector z (and state x) it reconstructs the original sample a.
Since computing the desired marginal $P_{\mathcal{B}}(A|X;\theta)$ is intractable, VAE algorithm optimizes a lower bound instead:

$$\log p(A|X;\theta) \geq \mathcal{L}(\theta,\phi;A|X) \tag{4.1}$$

$$\mathcal{L}(\theta,\phi;A|X) = \mathbb{E}_{q(z|A,X;\phi)}[\log p(A|z,X;\theta)] - D_{KL}(q(z|A,X;\phi) \mid\mid p(z;\theta)) \tag{4.2}$$

4.4 The O-DDACVaR

We introduce the Offline Distributional Deterministic Actor-Critic algorithm for CVaR optimization (O-DDACVaR), which is a fully offline variant of the DDACVaR algorithm able to learn effectively in the Batch RL setting.

We inspire ourselves on the BCQ algorithm presented in Fujimoto et al. (2019) to train a generative model G_w to generate actions $\hat{a}$ with high similarity to the dataset. Afterwards, we perturb $\hat{a}$ to generate actions a that maximize the CVaR of the return distribution. This perturbation is done by a perturbation model ξ, which is trained in the same way as the previously presented actor network for the DDACVaR algorithm, i.e. to maximize the CVaR of the estimated return distribution via deterministic policy gradients.

$$\hat{a_i} \sim G_w(x) \tag{4.3}$$

$$a_i = \hat{a_i} + \xi(x, \hat{a_i}; \phi, \theta^\xi) \tag{4.4}$$

where $\xi(s, a; \phi)$ is a perturbation (or actor) model that outputs an adjustment to action $\hat{a}$ in the range $[-\phi, \phi]$. ϕ is a fixed parameter which determines how much imitation learning is used for the final actions. Low values of ϕ imply small perturbations and hence the learnt policy will behave very similarly than the dataset policy, whereas higher values of ϕ will generate policies that behave more differently than the dataset policy but with a higher influence to maximize the CVaR. A good trade-off is crucial for the good performance of the algorithm.

Figure 4.1 shows a graphical representation of the O-DDACVaR algorithm with its 3 main parts: the generative model learning to generate actions similar to the ones in the dataset, the critic learning the value distribution and the actor learning the CVaR maximization policy by perturbing the actions generated by the generative model towards risk-sensitive ones.

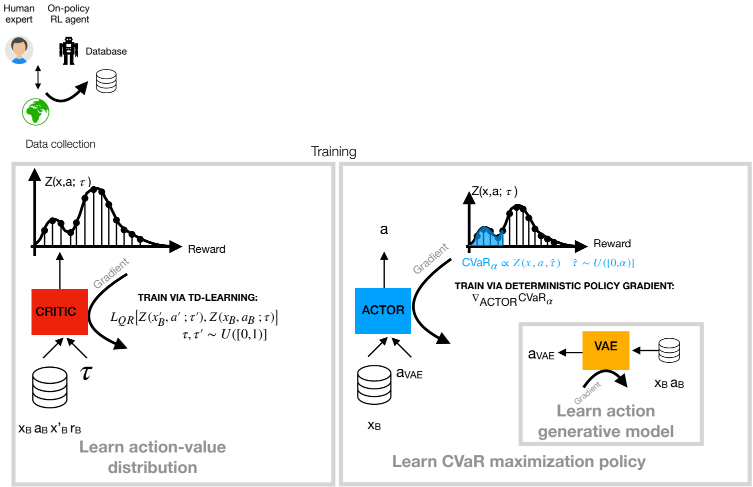
Human expert
On-policy RL agent
Database
Data collection
Training
$Z(x,a;\tau)$
Reward
Gradient
CRITIC
TRAIN VIA TD-LEARNING:
$L_{QR}[Z(x'_B, a'; \tau'), Z(x_B, a_B; \tau)]$
$\tau, \tau' \sim U([0,1)]$
τ
$x_B\ a_B\ x'_B\ r_B$
Learn action-value distribution
a
$Z(x,a;\tau)$
Reward
Gradient
$\mathrm{CVaR}_\alpha \propto Z(x, a, \hat{\tau})$ $\hat{\tau} \sim U([0,\alpha)]$
ACTOR
TRAIN VIA DETERMINISTIC POLICY GRADIENT:
$\nabla_{\mathrm{ACTOR}} \mathrm{CVaR}_\alpha$
a_{VAE}
x_B
a_{VAE}
VAE
$x_B\ a_B$
Gradient
Learn action generative model
Learn CVaR maximization policy

Figure 4.1: Graphical representation of the O-DDACVaR algorithm. Algorithm is implemented fully off-policy using an external training dataset obtained for example, by a human expert or an on-policy RL algorithm. Algorithm consists in 3 main blocks: the *VAE generative model*, the *actor* and the *critic*. The *VAE generative model* learns to generate actions similar to actions present in the training dataset. The *critic* learns the action value distribution via quantile regression TD-learning. The *actor* learns a policy to maximize the CVaR by perturbing the actions generated by the generative model towards risk-sensitive actions. It is trained using deterministic policy gradients of the CVaR computed by sampling from the estimated parameterized action-value distribution.

For our implementation, the prior $p(z;\theta)$ for the latent vector z is chosen to be a multivariate normal distribution $\mathcal{N}\big(0, Id\big)$, hence it doesn't require parameters θ. For the probabilistic encoders $q(z|a, x; \phi)$ we used a multivariate Gaussian with a diagonal covariance structure $\mathcal{N}(z|\mu(a, x; \phi), \sigma^2(a, x)Id)$, where μ and σ are parameterized by training parameter ϕ. To sample from the posterior $z_i \sim q(z|a, x; \phi)$ we use the reparameterization trick:

$$z_i = g(a_i, x_i, \epsilon; \phi) = \mu_i + \sigma_i \odot \epsilon \quad \epsilon \sim \mathcal{N}(0, Id) \tag{4.5}$$

where $\odot$ is the element-wise product.
For the decoder $p(a|z, x; \theta)$ we used a nonlinear deterministic function parameterized by θ.

When training the VAE, both recognition model parameters ϕ and the generative model parameters θ are learnt jointly to maximize the variational lower bound $\mathcal{L}(\theta, \phi; X)$ in 4.2 via gradient ascent, i.e. maximize expected reconstruction loss and a KL-divergence term according to the distribution of the latent vector z.
When both prior and posterior are Gaussian, KL-divergence can be computed analytically:

$$D_{KL}(q(z|X;\phi) \mid\mid p(z;\theta)) = -\frac{1}{2}\sum_{j=1}^{J}(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2) \tag{4.6}$$

where J is the dimensionality of z, and μ_j, σ_j represent the j-th element of the vectors.
The expected reconstruction error $\mathbb{E}_{q(z|X;\phi)}[\log p(X|z;\theta)]$ requires estimation by sampling. We use the mean-squared error between the action $a_{\mathcal{B}}$ from the dataset and the reconstructed action $\hat{a}$.

Finally, when generating actions during evaluation or deployment, random values of z are sampled from a multivariate normal and passed through the decoder to produce the actions. However, during deployment the stochasticity of the policy due to sampling the latent vector z from $z \sim \mathcal{N}(0, Id)$ can be removed by setting $\sigma = 0$.

The actor and the critic are trained in the same way as for the DDACVaR algorithm. The unique modification is that now the actor network has also as an input the action $\hat{a}$ generated by the generative model. For the sake of clarification it is trained by:

$$\nabla_{\theta^\xi} J_{\mathcal{B}}(\xi) \approx \mathbb{E}_{x\sim\rho^{\mathcal{B}}}\big[\nabla_{\theta^\xi}\xi(x, \hat{a})\nabla_a[\frac{1}{\alpha}\frac{1}{K}\sum_{i=1}^{K} Z(x, a; \tau_i)]|_{a=\xi(x,\hat{a})}\big] \tag{4.7}$$

Detailed information is provided in next section as well a pseudocode in 2.

4.5 Technical details of the algorithm

The algorithm works very similarly to the DDACVaR algorithm and hence almost all details are already described in section 3.3. Specifically, the critic is left untouched whereas the actor has some minor modifications which we will address below. With respect to the VAE, we use neural networks for both the probabilistic encoder and decoder. For the encoder $q(z|a, x; \phi)$ we use a neural network with a multivariate Gaussian output where μ and σ are the outputs of a neural network, i.e. nonlinear functions of datapoint (x_i, a_i) and ϕ. For the decoder $p(a|z, x; \theta)$ we use another neural network with deterministic output, i.e. nonlinear function of datapoint (x_i, z_i) and θ.

At every training-step a minibatch of observations $(x_t^B, a_t^B, r_t^B, x_{t+1}^B)$ is sampled from the static dataset $\mathcal{B}$. Given x_k^B, a_k^B from the dataset and the generative model G_w with its encoding E_w and decoding D_w parts, we sample actions $\hat{a}_k^{VAE}$ from the conditional VAE by:

$$\mu, \sigma, z = E_w(x_k^B, a_k^B) \quad \text{where } z = \mu + \sigma\mathcal{N}(0, Id) \tag{4.8}$$

$$\hat{a}_k^{VAE} = D_w(x_k^B, z) \tag{4.9}$$

Then we train jointly $[E_w, D_w]$ via gradient descent:

$$\mathcal{L}^{VAE}(E_w, D_w) = \sum_k [\hat{a_k}^{VAE} - a_k^B]^2 + 0.5D_{KL} \tag{4.10}$$

where D_{KL} is computed as in 4.6 using σ, μ from the outputs of the encoder.

With respect to the actor networks, we first remark that for batch RL, we do not need to account for exploration so there is no such behavior policy network.
The actor or *perturbation model* ξ is trained as in 4.7 where:

$$\hat{a} = D_w(x^B, z_{CLIP}) \quad z_{CLIP} = \text{clip}(z, -0.5, 0.5) \; z \sim \mathcal{N}(0, Id) \tag{4.11}$$

With respect to the critic network, we train it in the same way as for the DDACVaR algorithm. We also use target networks for the Bellman backup for both the critic and the actor ξ which are initialized like their counterparts, but constrained to slowly track the current networks so that to stabilize learning. With this goal, weights θ' of target networks are updated "softly" via: $\theta' \leftarrow (1-\tau)\theta' + \tau\theta$ where $\tau \in [0, 1], \tau \lll 1$.

Algorithm 2: O-DDACVaR

```
Input: Training dataset $\mathcal{B}$, quantile levels $N', N, K$, minibatch size $n$, $\alpha$
  confidence, $\Phi$ perturbation level
Initialize: Critic network $Z$, VAE network $G = \{E_G, D_G\}$ and actor network
  $\xi$ parameters and target networks $Z' \leftarrow Z$, $\xi' \leftarrow \xi$ parameters
while training do
    Sample minibatch of $n$ transitions $(x_{\mathcal{B}}, a_{\mathcal{B}}, x'_{\mathcal{B}}, r_{\mathcal{B}})$ from the dataset $\mathcal{B}$
    Train variational autoencoder:
     $\mu, \sigma = E_G(x_{\mathcal{B}}, a_{\mathcal{B}}) \quad \hat{a}^{VAE} = D_G(x_{\mathcal{B}}, z) \quad z = \mathcal{N}(\mu, \sigma)$
     $\theta^G \leftarrow \text{argmin}_{theta^G} \sum(a - \hat{a}^{VAE})^2 + \mathrm{D}_{KL}(\mathcal{N}(\mu, \sigma)||\mathcal{N}(0, 1))$
    Train critic:
    $\tau_i, \tau'_j \sim U([0, 1]) \quad 1 \leq i, j \leq N, N'$
    Compute next action a':
    $\hat{a}'^{VAE} \sim G_w(x'_{\mathcal{B}})$
    $a' = \hat{a}'^{VAE} + \xi'_{\Phi}(\hat{a}'^{VAE}, x'_{\mathcal{B}})$
    Compute distributional TD: $\delta_{ij} = r + \gamma Z'(x'_{\mathcal{B}}, a'; \tau'_j) - Z(x_{\mathcal{B}}, a_{\mathcal{B}}; \tau_i) \; \forall i, j$
    $\theta^Z \leftarrow \text{argmin}_{\theta_Z} \sum_{i=1}^{N} \mathbb{E}_{\tau'}[\rho_{\tau_i}(\delta_{ij})]$
    Train actor:
    $\tau_k \sim U([0, \alpha]) \quad 1 \leq k \leq K$
    $\hat{a}^{VAE} \sim G_w(x_{\mathcal{B}})$
    $a = \hat{a}^{VAE} + \xi_{\Phi}(\hat{a}^{VAE}, x_{\mathcal{B}})$
    Compute $\text{CVaR}_{\alpha} = \frac{1}{\alpha K} \sum_{k=1}^{K} Z(x_{\mathcal{B}}, a; \tau_k)$
    $\theta^{\xi} \leftarrow \text{argmax}_{\theta_{\xi}} CVaR_{\alpha}$
    Update target networks:
    $\theta^{Z'} \leftarrow \tau\theta^Z + (1 - \tau)\theta^{Z'}$; $\theta^{\xi'} \leftarrow \tau\theta^{\xi} + (1 - \tau)\theta^{\xi'}$
end
```

Chapter 5

Results

In this section we show the performance of our two algorithms (DDACVaR and O-DDACVaR) across a variety of continuous control tasks. First, we show DDACVaR performance in an off-policy setting on a toy example. Then, we move to the fully off-policy setting, i.e. batch RL setting, and we test O-DDACVaR performance on several external D4RL datasets, a recent suite of tasks and datasets for benchmarking progress in offline RL. (Fu et al., 2020). In all the results we compare the policies learnt when maximizing for the expected value (i.e. sampling $\tau \sim U([0,1])$, from now on *Mean*) and when maximizing for the α-CVaR (i.e. sampling $\tau \sim U([0,\alpha])$, from now on *CVaR*).

5.1 Off-policy setting

We test the performance of DDACVaR algorithm in an off-policy setting. We show that when the reward is deterministic the policies learnt by *Mean* and *CVaR* are the same. When adding uncertainty, we show how the policies differ to maximize their respective objective functions.

5.1.1 Car to goal

A car with fully-observable 2D-state: [position, velocity] must move to a goal position $x_F = 2.5\,\mathrm{m}$ starting at rest ($x_0 = 0.0\,\mathrm{m}, v_0 = 0.0\,\mathrm{m\,s^{-1}}$). The car can control its acceleration a which is constrained to range between [-1.0,1.0]$\mathrm{m\,s^{-2}}$ every $\varDelta k = 0.1$ seconds. Per every time step k before reaching the goal the car receives a penalization $R_k = -10$. If the car reaches the goal position, it receives a reward $R_F = +370$ and the episode ends. Otherwise, after $T_F = 400k$ the episode ends (with no extra penalization). We model the dynamics using a simple rectilinear uniform acceleration motion that updates the states of the agent at at every time step k:

$$v_{k+1} = v_k + a\varDelta k \tag{5.1}$$

$$x_{k+1} = x_k + v_k\varDelta k + 0.5a_k\left(\varDelta k\right)^2 \tag{5.2}$$

In the following, we test performance of the algorithm in 3 different settings. The first 2 settings, with no uncertainty, are meant to show that the algorithm is able to learn meaningful policies depending on the reward function used. The first setting has no extra modifications in the rewards and shows that for both *Mean* and *CVaR* the car learns to reach the goal in the shortest time possible. The second setting adds an extra penalization when velocity exceeds a certain value with probability 1. Since there is no uncertainty, in both *Mean* and *CVaR* the car learns to accelerate till the threshold velocity is reached and keeps without exceeding maximum velocity till the goal.

In the third setting we show how the 2 policies differ when penalization for high velocity occurs with low probability. *Mean* ignores the low probability penalizations and learns same policy as for setting 1, whereas *CVaR* evolves to a risk-sensitive policy and learns same policy as in setting 2 to ensure high penalizations do not occur.

5.1.2 Setting 1: No velocity penalization

The car arrives at the goal position with maximum acceleration throughout the whole episode for both *Mean* and *CVaR*. Car keeps an acceleration of $1\,\mathrm{m\,ts}^{-2}$ for the whole episode and hence reaches $x_F = 2.5$m in 24 time steps. Hence the final cumulative reward $G_T = (24 - 1)R_k + R_F = 140$.

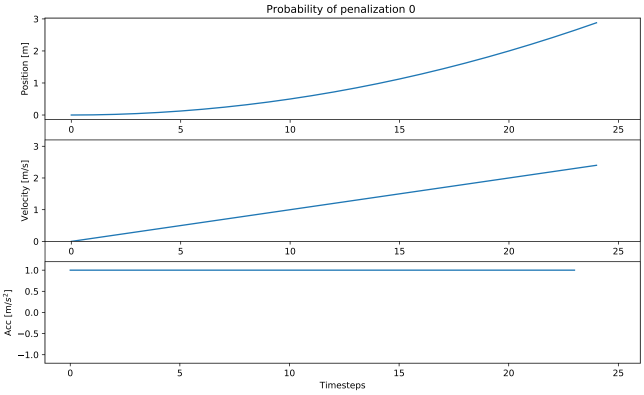

Figure 5.1: States and control evolution during evaluation without velocity penalization. The learnt policy is the same for both *Mean* and *CVaR*

5.1.3 Setting 2: Velocity penalization with probability 1

When the car velocity exceeds $1\mathrm{m\,ts}^{-1}$ it receives a penalization of $R_v = -25$. We expect both *CVaR* and *Mean* to learn similar policies since there is no reward uncertainty. As expected both learn to accelerate with maximum value until a velocity of $1\mathrm{m\,ts}^{-1}$ is reached and then they keep with $1\mathrm{m\,ts}^{-1}$ velocity until the goal.

Starting from $x_0 = 0$m, the car reaches a velocity of $1\mathrm{m\,ts}^{-1}$ after 10 time steps, at $x_{k=10} = 0.5$. Keeping velocity $1\mathrm{m\,ts}^{-1}$ through the rest of the episode, it reaches the goal position after 21 time steps, i.e. in a total of 31 steps. Hence the final cumulative reward $G_T = (31 - 1)R_k + R_F = 70$.

We notice that the reward values were deliberately chosen in order to ensure that for the Setting 2, driving faster than $1\,\mathrm{m\,ts}^{-1}$ never induces higher cumulative rewards. In that hypothetic case, car will reach goal in 24 time steps but would add penalization for driving over the speed limit for 10 time steps:
$G_T = (24-1)R_k + (21-1)R_v + R_F << (31-1)R_k + R_F = 70$

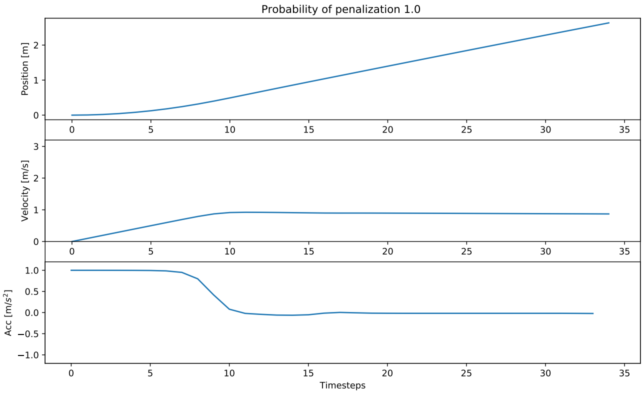

Figure 5.2: States and control evolution during evaluation with velocity penalization with probability 1. The learnt policy is the same for both *Mean* and *CVaR*

5.1.4 Case 3: Velocity penalization with probability P=0.2

When penalization only occurs with a certain low probability P=0.2, we can notice a difference in the policies learnt by *CVaR* and *Mean*. *Mean* learns policies so that it maximizes the expected value by keeping a linear increase of the velocity during the whole episode disregarding the low probability events with high penalizations, whereas *CVaR* saturates the velocity, even though the probability of a penalization is low.

In the following we show the evaluation during training for both *Mean* and *CVaR*. Specifically, every 10 training episodes we test the current policy over 500 evaluation episodes. During evaluation, we remove the exploration noise from the policy. During this 500 evaluation episodes, we track the mean of the cumulative rewards (see Figure 5.3), the CVaR with a confidence level of 0.1 (see Figure 5.4) and also the mean of the number of time steps driving over the speed limit (see Figure 5.5). We can see how *Mean* converges to slightly higher expected value but lower CVaR than *CVaR*. Additionally, under *Mean* the car learns to drive over the speed limit for a long amount of time while *CVaR* learns not to exceed it already in the very early stages of learning.
The Figures also show that *Mean* takes longer to converge than *CVaR*, with a very high standard deviation until the very end due to some episodes in the evaluation process for which the car doesn't reach the goal and keeps accumulating R_k penalizations. This suggests that in some scenarios the fact of reducing high penalization outcomes, even though they occur with very low probability, might reduce training time.

After training we used the final policies and deployed them in the environment. Obtained trajectories are shown in Figure 5.6. A histogram of the resulting cumulative rewards shows how the *Mean* algorithm obtains, in average, higher cumulative rewards, but in some scenarios it obtains really low results, whereas the *CVaR* performs in average worse than *Mean* but the conditional value-at-risk is higher than for the *Mean* (see Figure 5.7)

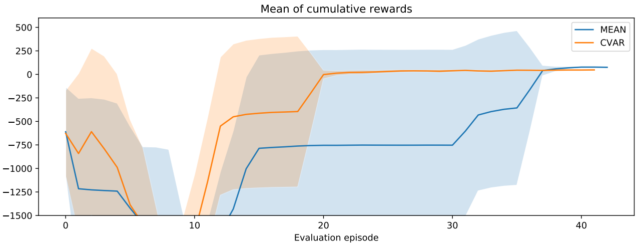

Figure 5.3: Evolution during training of mean of the cumulative rewards over 500 evaluation episodes. Every datapoint corresponds to one evaluation process performed every 10 training episodes. For the plot we show averaged values over 5 random seeds and 1 standard deviation

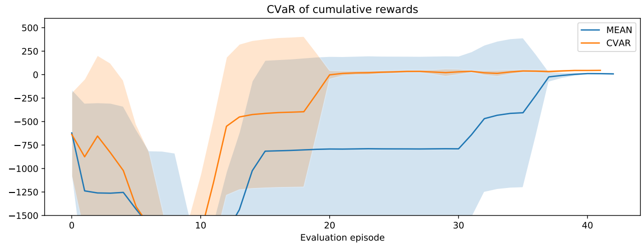

Figure 5.4: Evolution during training of $\text{CVaR}_{\alpha=0.1}$ of the cumulative rewards over 500 evaluation episodes. Every datapoint corresponds to one evaluation process performed every 10 training episodes. For the plot we show averaged values over 5 random seeds and 1 standard deviation

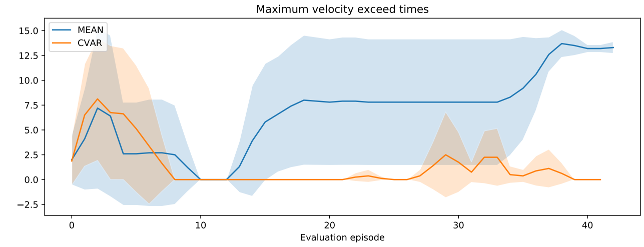

Figure 5.5: Evolution during training of mean of number of time steps driving over speed limit. Every datapoint corresponds to one evaluation process performed every 10 training episodes. For the plot we show averaged values over 5 random seeds and 1 standard deviation

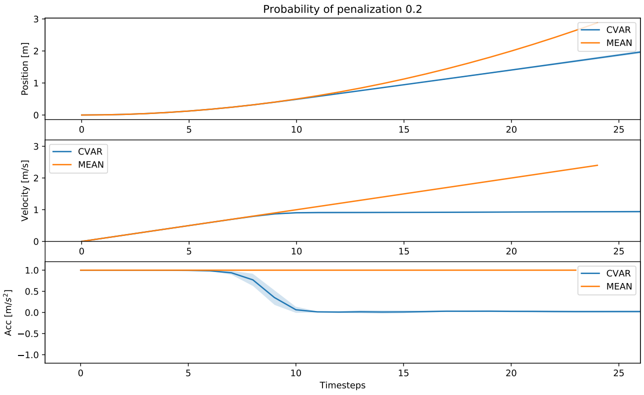

Figure 5.6: States and control evolution during evaluation with velocity penalization with probability 0.2. The learnt policies by *Mean* and *CVaR* differ, *CVaR* leading towards risk-averse behavior

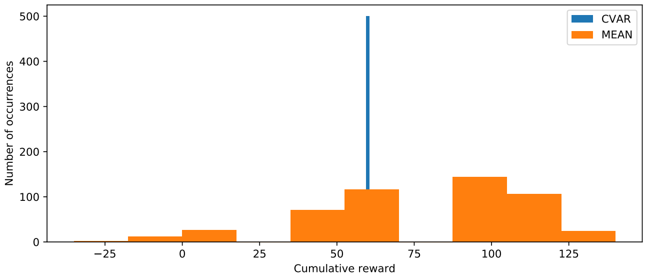

Figure 5.7: Comparison of cumulative rewards achieved with *CVaR* and *Mean* when the probability of velocity penalization is 0.2. *Mean* achieves a higher expected value ($\mu = 78.1$) but has a lower CVaR (CVaR$_{\alpha=0.1}$=12) compared to *CVaR* ($\mu = 60$ and CVaR$_{\alpha=0.1}$=60). The data was obtained by acting for 500 episodes with the final policies after training. The distribution of rewards for *Mean* follows a binomial distribution B(n,p) with n being the number of time steps over the speed limit and p the probability of penalization.

5.2 Batch RL Setting

We test the performance of the O-DDACVaR algorithm in a Batch RL setting. To this end, we make use of the D4RL datasets, specifically datasets from D4RL generated on two different physical control tasks from the Gym toolkit developed under MuJoCo physics simulator (Todorov et al., 2012): *HalfCheetah* and *Walker2d*.

5.2.1 *HalfCheetah*

We show results on the *HalfCheetah* task. In the *HalfCheetah* environment, a 2D cheetah robot needs to learn to run. The observation space has 17 dimensions, accounting for velocity and position of the joints. The control space has a dimensionality of 6, constrained between -1 and 1, controlling the torques applied to the joints. The reward received every time step is a function of the control cost and the velocity of the agent.

Dataset
We use the *halfcheetah-medium-v0* dataset from D4RL. This dataset consists of 1M samples of data collected by using a partially trained policy. Specifically, they train a policy using a standard online RL algorithm (soft actor-critic) until it reaches approximately 1/3 the performance of the expert.
The environment is deterministic so to proof the performance of our algorithm we modify the environment to add a source of uncertainty. We introduce stochasticity in the original cost function in a way that makes the environment stochastic enough to have a meaningful assessment of risk in terms of tail performance. If the velocity of the agent is greater than 4, a reward of $R_v = -100$ is given with probability 0.05 Due to the fact that there is no direct access to the velocity of the cheetah from the observations, we reran the environment interactions from the dataset in an open-loop way to collect the desired information and modify the rewards accordingly. The modified dataset was used to train the algorithm in an offline way. Only for the evaluation process, we allowed the agent to interact with the environment. We use the *HalfCheetah-v3* environment from the OpenAI Gym Toolkit, but modified with a *RewardWrapper* to change the default reward function.

In the following we show the evaluation during training for both *Mean* and *CVaR*. Specifically, every 1000 training steps we evaluate the current policy for 5 episodes. During evaluation, to ensure the deterministic nature of the policy the latent vector z is not sampled from a Gaussian, but a vector of zeros is used instead. During this 5 evaluation episodes, we track the mean of the cumulative rewards (see Figure 5.8), the CVaR with a confidence level of 0.1 (see Figure 5.9) and also the mean of number of time steps running over the speed limit (see Figure 5.10).
We can see how *Mean* converges to slightly higher expected value but far lower conditional value-at-risk than *CVaR*. Additionally, under *Mean* the cheetah runs over the speed limit in many times while *CVaR* learns not to exceed it already in the very early stages of learning.

After training we used the final policies and deployed them in the noisy environment for 500 episodes 200 time steps each. A histogram of the resulting cumulative rewards shows how the *Mean* algorithm obtains, in average, higher rewards, but in some scenarios it obtains really low results, whereas the *CVaR* performs in average worse than *Mean* but the conditional value-at-risk is higher than for the *Mean* (see

Figure 5.11).

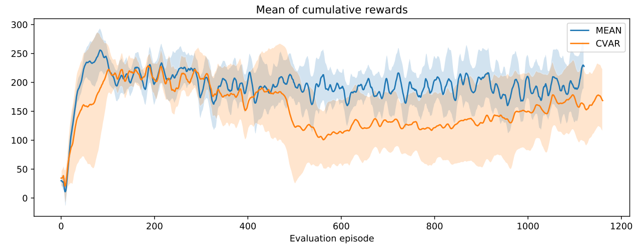

Figure 5.8: Evolution during training of the mean of the cumulative rewards over 5 evaluation episodes. Every point corresponds to one evaluation process performed every 1000 steps of training. For the plot we show averaged values over 5 random seeds and 1 standard deviation

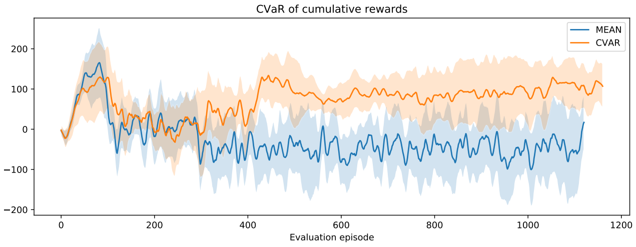

Figure 5.9: Evolution during training of $\mathrm{CVaR}_{\alpha=0.1}$ of the cumulative rewards over 5 evaluation episodes. Every point corresponds to one evaluation process performed every 1000 steps of training. For the plot we show averaged values over 5 random seeds and 1 standard deviation

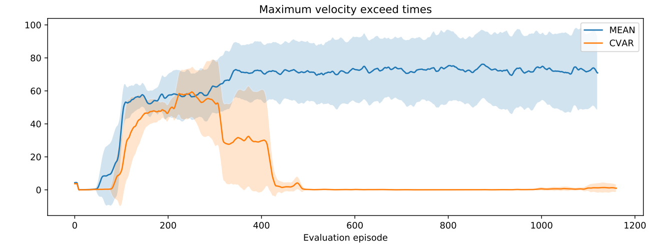

Figure 5.10: Evolution during training of the mean of number of times running over speed limit over 5 evaluation episodes. Every point corresponds to one evaluation process performed every 1000 steps of training. For the plot we show averaged values over 5 random seeds and 1 standard deviation

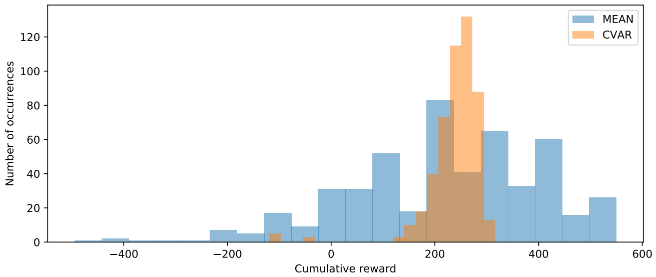

Figure 5.11: Comparison of cumulative rewards achieved with *CVaR* and *Mean*. The data was obtained by acting for 500 episodes of 200 time steps each with the final policies after training. *Mean* achieves an expected value (μ = 220.75) but has a far lower conditional value-at-risk (CVaR$_{\alpha=0.1}$=-131.77) compared to *CVaR* (μ = 238.18 and CVaR$_{\alpha=0.1}$=130.63)

5.2.2 *Walker2d*

We show results on the *Walker2d* task. In the *Walker2d* environment, a two-dimensional bipedal robot needs to learn to walk forward as fast as possible. The observation space has 17 dimensions (accounting for velocity and position of the joints). The control space has a dimensionality of 6, constrained between -1 and 1, controlling the torques applied to the joints. The reward received every time step is a function of the control cost, the velocity of the agent and and an additional reward every time step the agent keep in a *healthy* position. A position is considered to be *healthy* when the agent has an angle inclination and center of mass position constrained between some pre-defined values. Whenever this doesn't hold, the episode ends.

Dataset
We use the *walker2d-expert-v0* dataset from D4RL which consists of 1M samples of data generated by training to completion a policy online using a standard RL algorithm (soft actor-critic).
The environment is deterministic so to proof the performance of our algorithm we modify the environment to add a source of uncertainty. We introduce stochasticity in the original cost function in a way that makes the environment stochastic enough to have a meaningful assessment of risk in terms of tail performance. In this case, we specify a *robust healthy* range, a subset of *healthy* range. A reward of $R_H = -100$ is given with probability 0.1, if the walker exits the *robust healthy* space.

Due to the fact that there is no direct access to the required information from the observations, we reran the environment interactions from the dataset in an open-loop way to collect the desired information and modify the rewards accordingly. The modified dataset was used to train the algorithm in an offline way. Only for the evaluation process, we allowed the agent to interact with the environment. We use the *Walker2d-v3* environment from the OpenAI Gym Toolkit, but modified with a *RewardWrapper* to change the default reward function.

In the following we show the evaluation during training for both *Mean* and *CVaR*. Specifically, every 1000 training steps we evaluate the current policy for 5 episodes. During evaluation, to ensure the deterministic nature of the policy the latent vector z is not sampled from a Gaussian, but a vector of zeros is used instead. During this 5 evaluation episodes, we track the mean of the cumulative rewards (see Figure 5.12), the CVaR with a confidence level of 0.1 (see Figure 5.13) and also the mean of number of time steps where the cheetah orientation doesn't lie in the robust healthy space (see Figure 5.14).
In this case, we can see that *CVaR* significantly outperforms *Mean* in both expected value and conditional value-at-risk and this happens at all stages of training. Regarding the number of times the agent orientation doesn't lie in the robust healthy space, we cannot see much difference during early stages of training but at the end, *CVaR* shows a clear decrease, although it is not that substantial. This outperformance could be explained by the fact that *CVaR*, preventing low probability penalizations (i.e. preventing the walker to exit the robust healthy space), could help subsequently in preventing the walker to fall forward or backwards. Hence, leading to better policies with respect to both mean and conditional value-at-risk.

After training we used the final policies and deployed them in the noisy environment for 500 episodes 1000 time steps each. A histogram of the resulting cumulative rewards shows how the *CVaR* algorithm outperfoms *Mean* in both expected value and conditional value-at-risk, although differences are not that substantial. 5.15)

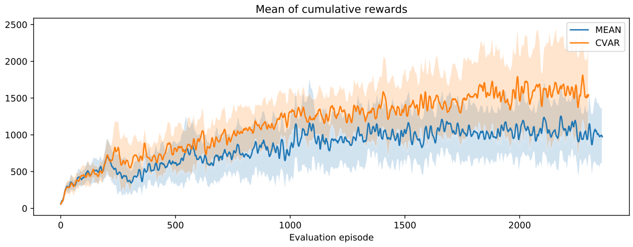

Figure 5.12: Evolution during training of the mean of the cumulative rewards over 5 evaluation episodes. Every point corresponds to one evaluation process performed every 1000 steps of training.For the plot we show averaged values over 10 random seeds and 1 standard deviation

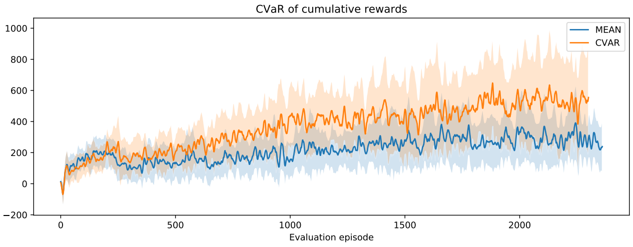

Figure 5.13: Evolution during training of $\mathrm{CVaR}_{\alpha=0.1}$ of the cumulative rewards over 5 evaluation episodes. Every point corresponds to one evaluation process performed every 1000 steps of training.For the plot we show averaged values over 10 random seeds and 1 standard deviation

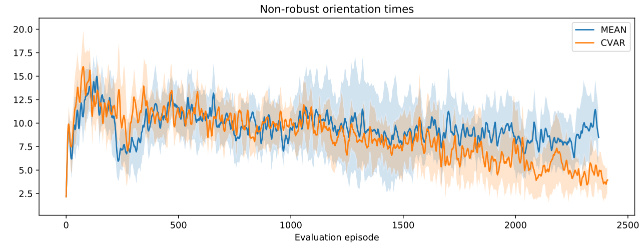

Figure 5.14: Evolution during training of the mean of number of times in a 'non-robust healthy' orientation over 5 evaluation episodes. Every point corresponds to one evaluation process performed every 1000 steps of training. For the plot we show averaged values over 5 random seeds and 1 standard deviation

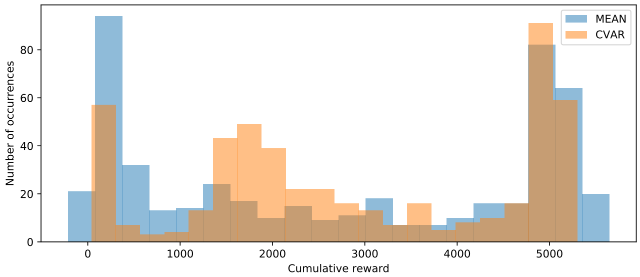

Figure 5.15: Comparison of cumulative rewards achieved with *CVaR* and *Mean* over 500 evaluation episodes of 1000 time steps each using the trained final policies. *Mean* achieves a slightly lower expected value ($\mu = 2739.47$) and has a lower conditional value-at-risk ($\text{CVaR}_{\alpha=0.1}$=67.54) compared to *CVaR* ($\mu = 2922.56$ and $\text{CVaR}_{\alpha=0.1}$=235.74)

5.2.3 Medical environment

Add more environments. To be decided

Chapter 6

Discussion

In this thesis we presented DDACVaR and O-DDACVaR , two novel actor-critic frameworks to learn risk-sensitive policies by maximizing CVaR. O-DDACVaR algorithm can be implemented fully off-policy, hence it can learn from a fixed dataset without further interaction with the environment, which is crucial for tasks where the data collection procedure is costly, time-consuming and, most importantly for our goal, when it is risky.

By using a distributional RL approach to learn the full value distribution, our algorithm has the ability to learn policies with different level of risk-aversity depending on the confidence level selected. Additionally, the fact of being able to switch the objective function just by changing the sampling distribution for τ gives to the algorithm a lot of versatility. Algorithms that directly derive policies for CVaR optimization (Chow and Ghavamzadeh, 2014; Tamar et al., 2015) don't show such property. In the future, we could explore the range of policies that we can learn by playing with the sampling distribution.

We tested performance in several environments and showed difference in learnt policies depending on the chosen confidence level, ones maximizing the mean of the return to the detriment of rare high penalizations happening and the others, reducing expected value but ensuring no high penalizations occur.
When testing for complex robotics Mujoco environments, we actually can see how the fact of maximizing for the CVaR can accelerate training and reach overall better performances than when maximizing for the expected value.
A more detailed analysis of the effect of the confidence level on training times and final performance could be done in the future, as well as, the effect on the maximal perturbation level allowed to the actor network on the VAE generated actions.
Additionally, we could test the algorithm in other environments such as for autonomous navigation or healthcare applications where a lot of offline data is available and where catastrophic events prevention is crucial.

The offline capability of O-DDACVaR holds tremendous promise for making possible to effectively allowing to turn any large enough dataset into a risk-sensitive policy without the need of exposing the agent to the risky environment during the training process. Additionally, we consider the algorithm can serve as a base for future algorithms that not only scale to real-world problems, but will also lead to solutions that generalize substantially better. We believe O-DDACVaR looks promising for advancing in the safe and generalizable data-driven RL area.

Bibliography

Y. C. Tang, J. Zhang, and R. Salakhutdinov, "Worst Cases Policy Gradients," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 2020, pp. 1078–1093.

J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," 2015.

S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a Formal Model of Safe and Scalable Self-driving Cars," *CoRR*, vol. abs/1708.0, 2017.

P. Artzner, F. Delbaen, J. M. Eber, and D. Heath, "Coherent measures of risk," *Mathematical Finance*, 1999.

R. T. Rockafellar and S. Uryasev, "Optimization of conditional value-at-risk," *The Journal of Risk*, vol. 2, no. 3, pp. 21–41, 2000.

A. Majumdar and M. Pavone, "How Should a Robot Assess Risk? Towards an Axiomatic Theory of Risk in Robotics," 2020.

J. A. Bagnell, A. Y. Ng, and J. G. Schneider, "Solving Uncertain Markov Decision Processes," *Carnegie Mellon Research Showcase*, 2001.

J. Morimoto and K. Doya, "Robust reinforcement learning," *Neural Computation*, 2005.

L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *34th International Conference on Machine Learning, ICML 2017*, 2017.

M. Heger, "Consideration of Risk in Reinforcement Learning," in *Machine Learning Proceedings 1994*, 1994.

S. P. Coraluppi and S. I. Marcus, "Mixed risk-neutral/minimax control of discrete-time, finite-state Markov decision processes," *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 528–532, 2000.

S. Coraluppi and S. I. Marcus, "Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes," *Automatica*, vol. 35, no. 2, pp. 301–309, 1999.

E. Altman, "Asymptotic properties of constrained Markov Decision Processes," *ZOR - Methods and Models of Operations Research*, 1993.

P. Geibel, "Reinforcement Learning for MDPs with Constraints," in *Machine Learning: ECML 2006*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 646–653.

A. Tamar, D. Di Castro, and S. Mannor, "Policy gradients with variance related risk criteria," in *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 2012.

R. A. Howard and J. E. Matheson, "Risk-Sensitive Markov Decision Processes," *Management Science*, vol. 18, no. 7, pp. 356–369, 1972.

K. J. Chung and M. J. Sobel, "Discounted MDPs: Distribution functions and exponential utility maximization," *SIAM Journal on Control and Optimization*, 1987.

M. Sato, H. Kimura, and S. Kobayashi, "TD algorithm for the variance of return and mean-variance reinforcement learning," *Transactions of the Japanese Society for Artificial Intelligence*, 2001.

Y. Chow, A. Tamar, S. Mannor, and M. Pavone, "Risk-sensitive and robust decision-making: A CVaR optimization approach," *Advances in Neural Information Processing Systems*, vol. 2015-Janua, pp. 1522–1530, 2015.

M. Petrik and D. Subramanian, "An approximate solution method for large risk-averse markov decision processes," in *Uncertainty in Artificial Intelligence - Proceedings of the 28th Conference, UAI 2012*, 2012.

T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka, "Parametric return density estimation for Reinforcement Learning," in *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence, UAI 2010*, 2010.

T. Morimura, Tetsuro and Sugiyama, Masashi and Kashima, Hisashi and Hachiya, Hirotaka and Tanaka, "Nonparametric return distribution approximation for reinforcement learning," in *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*, 2010.

A. Tamar, Y. Glassner, and S. Mannor, "Optimizing the CVaR via sampling," *Proceedings of the National Conference on Artificial Intelligence*, vol. 4, pp. 2993–2999, 2015.

Y. Chow and M. Ghavamzadeh, "Algorithms for CVaR optimization in MDPs," *Advances in Neural Information Processing Systems*, vol. 4, no. January, pp. 3509–3517, 2014.

M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *34th International Conference on Machine Learning, ICML 2017*, 2017.

W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018.

W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *35th International Conference on Machine Learning, ICML 2018*, 2018.

M. A. Taleghan and T. G. Dietterich, "Efficient Exploration for Constrained MDPs," in *AAAI Spring Symposia*, 2018.

D. Silver, G. Lever, D. Technologies, G. U. Y. Lever, and U. C. L. Ac, "Deterministic Policy Gradient (DPG)," *Proceedings of the 31st International Conference on Machine Learning*, 2014.

G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, T. B. Dhruva, A. Muldal, N. M. O. Heess, and T. P. Lillicrap, "Distributed Distributional Deterministic Policy Gradients," *ArXiv*, vol. abs/1804.0, 2018.

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

E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IEEE International Conference on Intelligent Robots and Systems*, 2012.

R. Sutton and A. Barto, "Reinforcement Learning: An Introduction," *IEEE Transactions on Neural Networks*, 1998.

D. Bertsekas, "Nonlinear Programming." *SIAM AMS Proc*, vol. 9, 1976.

C. Acerbi and D. Tasche, "On the coherence of expected shortfall," *Journal of Banking and Finance*, 2002.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, 1992.

M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, 2004.

T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," in *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 2012.

R. Koenker, *Quantile Regression*, ser. Econometric Society Monographs. Cambridge University Press, 2005.

P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, 1964.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, 2015.

G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Physical Review*, 1930.

S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *35th International Conference on Machine Learning, ICML 2018*, 2018.

D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

S. S. Wang, "A Class of Distortion Operators for Pricing Financial and Insurance Risks," *The Journal of Risk and Insurance*, 2000.

S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems," 2020.

T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft Actor-Critic Algorithms and Applications," *CoRR*, vol. abs/1812.0, 2018.

M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning," *CoRR*, vol. abs/1710.0, 2017.

S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *36th International Conference on Machine Learning, ICML 2019*, 2019.

A. Kumar, J. Fu, G. Tucker, and S. Levine, "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction," no. NeurIPS, 2019.

D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

Appendix A

A.1 Distributional RL

In contrast to standard RL where we model the expected return, in distributional RL we aim to model the full distribution of the return, i.e the *value distribution.* Bellemare et al. (2017); Dabney et al. (2018a,b) presented different algorithms for distributional RL and they argued the advantages of learning the whole distribution even when we want to optimize the expected value, since the distribution can give us more information that can be beneficial to learn function approximations. Its advantages become more obvious when designing risk-sensitive algorithms that aim to reduce probability of certain outcomes.

A.1.1 Distributional Bellman Operator $\mathcal{T}^\pi$

We define the random return $Z^\pi(x, a)$ as the random variable that represents the sum of discounted rewards obtained by starting from position x taking action a and thereupon following policy π. This variable captures intrinsic randomness from: immediate rewards, stochastic dynamics and possibly an stochastic policy. It follows:

$$Q^\pi(x, a) = \mathbb{E}[Z^\pi(x, a)] \tag{A.1}$$

Similarly than Q-value 3.2, Z is also described by a recursive equation but of a distributional nature:

$$Z^\pi(x, a) \overset{D}{=} R(x, a) + \gamma Z(x', a') \tag{A.2}$$

where $x' \backsim p(\cdot|x, a)$ and $a' \backsim \pi(\cdot|x')$ and $\overset{D}{=}$ denotes that the random variables on both sides of the equation share the same probability distribution.
The *distributional Bellman equation* defined in (A.2) states that the distribution of Z is characterized by the interaction of 3 random variables, the random reward R, the next state-action (x',a') and its random return Z(x',a'). From here on, we will view Z^π as a mapping from state-action pairs to distributions over returns, and we call this distribution the *action-value distribution.*

In the policy evaluation setting (Sutton and Barto, 1998) one aims to find the value $V^\pi(x)$ or $Q^\pi(x, a)$ function associated with a given fixed policy π. In the distributional case we aim to find Z^π.

We view the reward function as a random vector R $\in \mathbb{Z}$ and define the transition operator $P^{\pi} : \mathbb{Z} \to \mathbb{Z}$

$$P^{\pi} Z(x,a) \overset{D}{=} Z(X', A') \tag{A.3}$$

$$X' \backsim\ P(\cdot|x,a) \text{ and } A' \backsim\ \pi(\cdot|X') \tag{A.4}$$

where we use capital letters to emphasize the random nature of the next state-action pair (X',A').
Bellemare et al. (2017) defined the Distributional Bellman operator T^{π} as:

$$T^{\pi} Z(x,a) \overset{D}{=} R(x,a) + \gamma P^{\pi} Z(x,a) \tag{A.5}$$

Bellemare et al. (2017) showed that (A.5) is a contraction mapping in Wasserstein metric whose unique fixed point is the random return Z^{π}.

Wasserstein metric:

The p-Wasserstein metric W_p for $p \in [1, \infty]$, also known as the Earth Mover's Distance when $p = 1$, is an integral probability metric between distributions. The p-Wasserstein distance is characterized as the L^p metric on inverse cumulative distribution functions. That is, the p-Wasserstein metric between distributions U and Y is given by:

$$W_p(U, Y) = \big(\int_0^1 |F_Y^{-1}(w) - F_U^{-1}(w)|^p dw \big)^{\frac{1}{p}} \tag{A.6}$$

where for a random variable Y, the inverse CDF F_Y^{-1} of Y is defined by:

$$F_Y^{-1}(w) \coloneqq \inf\big\{ y \in \mathbb{R} \; | w \ \leq F_Y(w) \big\} \tag{A.7}$$

where $F_Y(w) = Pr(y \leq Y)$.
A.6 shows that the Wasserstein metric is the integral over the difference in the quantile functions of the random variables U and Y. A.1 illustrates the 1-Wasserstein distance as the area between two cumulative distribution functions. Unlike the Kullback-Leibler divergence, the Wasserstein metric is a true probability metric and considers both the probability of and the distance between various outcome events, which makes it well-suited to domains where an underlying similarity in outcome is more important than exactly matching likelihoods.

Contraction in $\hat{d}_p$:

Consider the process $Z_{k+1} \coloneqq T^{\pi} Z_k$, starting with some $Z_0 \in \mathcal{Z}$, where $\mathcal{Z}$ is the space of action-value distributions with bounded moments.
Bellemare et al. (2017) showed that $T^{\pi} Z : \mathcal{Z} \to \mathcal{Z}$ is a γ-contraction in the maximal form of the Wasserstein metric $\hat{d}_p$ defined as

$$\hat{d}_p(Z_1, Z_2) \coloneqq \sup_{x,a} W_p(Z_1(x,a), Z_2(x,a)) \tag{A.8}$$

i.e. for any two $Z_1, Z_2 \in \mathcal{Z}$,

$$\hat{d}_p(T^{\pi} Z_1, T^{\pi} Z_2) \leq \gamma \hat{d}_p(Z_1, Z_2) \tag{A.9}$$

Using Banach's fixed point theorem, we can conclude that T^{π} has a unique fixed point, which by inspection must be Z^{π}.

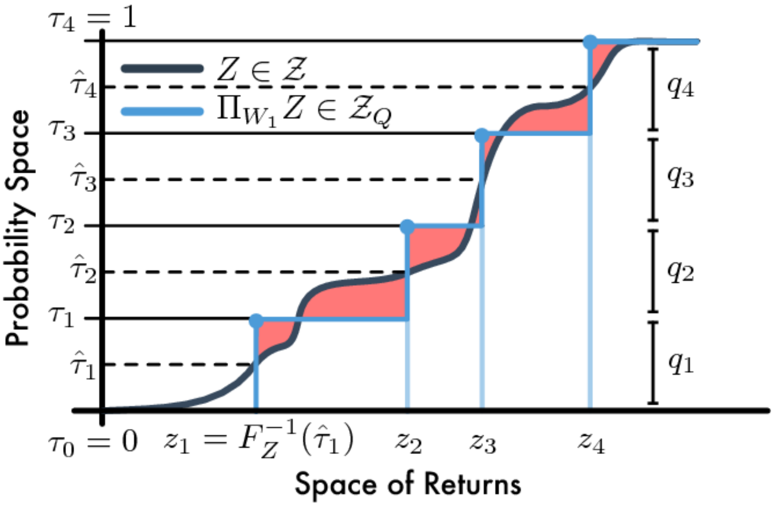

Figure A.1: Illustration of 1-Wasserstein distance as the area between 2 cumulative distribution functions. Shaded regions sum to form the 1-Wasserstein error. Source: Dabney et al. (2018a)

Hence the $\hat{d}_p$ metric is shown to be a useful metric for studying behavior of distributional RL algorithms and to show their convergence to a fixed point.
Moreover, Bellemare et al. (2017) showed that $\hat{d}_p$ can be used to learn a value distribution by minimizing the Wasserstein distance between a distribution Z and its distributional Bellman update $T^\pi Z$, analogously to the way that TD-learning attempts to iteratively minimize the L^2 distance between Q and TQ.

A.1.2 Distributional Bellman Optimality Operator $\mathcal{T}$

We have so far considered a policy evaluation setting and we studied the behavior of its associated distributional operator T^π. We now move to the control setting where we aim to find a policy π^* that optimizes the value function and the corresponding distribution.

While all optimal policies attain the same value Q^*, in general there are many optimal value distributions. We will call a *distributional Bellman optimality operator* any operator $\mathcal{T}$ which implements a greedy selection rule, i.e.:

$$\mathcal{T}Z = \left\{\mathcal{T}^\pi Z \text{ for some } \pi \in \mathcal{G}_\mathcal{Z}\right\}$$

where $\mathcal{G}_\mathcal{Z}$ is the set of greedy policies for Z, i.e. set of policies that maximize the expectation of Z.

As in the policy evaluation setting, we are interested in the behavior of the iterates $Z_{k+1} := \mathcal{T}Z_k, Z_0 \in \mathcal{Z}$. Bellemare et al. (2017) shows that the distributional analogue of the Bellman optimality operator converges, in a weak sense, to the set of optimal value distributions, but this operator is *not a contraction in any metric between distributions.*
Another result from Bellemare et al. (2017) shows that we cannot in general minimize the Wasserstein metric, viewed as a loss, using stochastic gradient descent methods. This limitation, is crucial in a practical context when the value distribution needs to be approximated.

A.1.3 Quantile approximation

Dabney et al. (2018a) used the theory of quantile regression Koenker (2005) to design an algorithm applicable in a stochastic approximation setting. Quantile regression is used to estimate the quantile function at precisely chosen points. Then the Bellman update is applied onto this parameterized quantile distribution. This combined operator is proven to be a contraction and the estimated quantile function is shown to converge to the true value distribution when minimized using stochastic approximation.

Quantile projection

We aim to estimate the quantiles of the value distribution, i.e. the values of the return that divide the value distribution in equally sized parts. We call it quantile distribution and we let $\mathcal{Z}_Q$ be the space of quantile distributions. We denote the cumulative probabilities associated with such a distribution, known as quantile levels, by $\tau_1, \tau_2 ... \tau_N$, so that $\tau_i = \frac{i}{N}$ for $i = 1, ...N$.
Formally, let $\theta : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}^N$ be some parametric model. A quantile distribution $Z_\theta \in \mathcal{Z}_Q$ maps each state-action pair (x,a) to a uniform probability distribution supported on $\{\theta_i(x,a)\}$. Hence we can approximate it by a uniform mixture of N Diracs:

$$Z_\theta(x,a) \coloneqq \frac{1}{N} \sum_{i=1}^{N} \delta_{\theta_i(x,a)} \tag{A.10}$$

with each θ_i assigned a fixed quantile.
We aim to learn the support of these Diracs, i.e. learn θ_i $\forall (i,a,x)$ triplet. We will do it by quantifying the projection of an arbitrary value distribution $Z \in \mathcal{Z}$ onto $\mathcal{Z}_Q$, that is:

$$\prod_{W_1} Z \coloneqq \underset{Z_\theta \in \mathcal{Z}_Q}{\arg\min} W_1(Z, Z_\theta) \tag{A.11}$$

This projection $\prod_{W_1}$ is the quantile projection. We can quantify the projection between distribution Y and U, a uniform distribution over N Diracs as in (A.10) with support $\{\theta_1, ...\theta_N\}$, by:

$$W_1(Y,U) = \sum_{i=1}^{N} \int_{\tau_{i-1}}^{\tau_i} |F_Y^{-1}(w) - \theta_i| dw \tag{A.12}$$

Lemma 2 in Dabney et al. (2018a) shows that the set of values $\{\theta_1, ..., \theta_N\}$ that minimize $W_1(Y,U)$ are given by $\theta_i = F_Y^{-1}(\hat{\tau}_i)$, where $\hat{\tau}_i = \frac{\tau_{i-1}+\tau_i}{2}$. Figure A.1 shows the projection $\mathit{\Pi}_{W_1} Z$ of the function Z onto $\mathcal{Z}_Q$ minimizing the 1-Wasserstein distance to Z.

A.1.4 Quantile Regression

Quantile regression is a method for approximating quantile functions of a distribution at specific points. The quantile regression loss for quantile $\tau \in [0,1]$ is an asymmetric convex lox function that penalizes underestimation errors with weight τ and overestimation errors with weight $1 - \tau$.

For a distribution Z and given quantile τ, the value of the quantile function $F_Z^{-1}(\tau)$ may be characterized as the minimizer of the quantile regression loss:

$$\mathcal{L}_{QR}^{\tau}(\theta) = \mathbb{E}_{\hat{Z}\backsim Z}[\rho_\tau(\hat{Z}-\theta)] \tag{A.13}$$
$$\rho_\tau(u) = u(\tau - \mathbb{1}_{\{u<0\}}) \quad \forall u \in \mathbb{R}$$

Given that the minimizer of the quantile regression loss for τ is $F_Z^{-1}(\tau)$ and using Lemma 2 in Dabney et al. (2018a) (A.1.3) that claims that the values of $\{\theta_1, ...\theta_N\}$ that minimize $W_1(Z, Z_\theta)$ are given by $\theta_i = F_Y^{-}1(\hat{\tau}_i)$, we have that the set of minimizing values $\{\theta_1, ...\theta_N\}$ for $W_1(Z, Z_\theta)$ are the minimizers of the following objective:

$$\sum_{i=1}^{N} \mathbb{E}_{\hat{Z}\sim Z}[\rho_{\hat{\tau}_i}(\hat{Z}-\theta_i)] \tag{A.14}$$

This loss gives unbiased sample gradients and hence, we can find the minimizing $\{\theta_1, ...\theta_N\}$ by stochastic gradient descent.

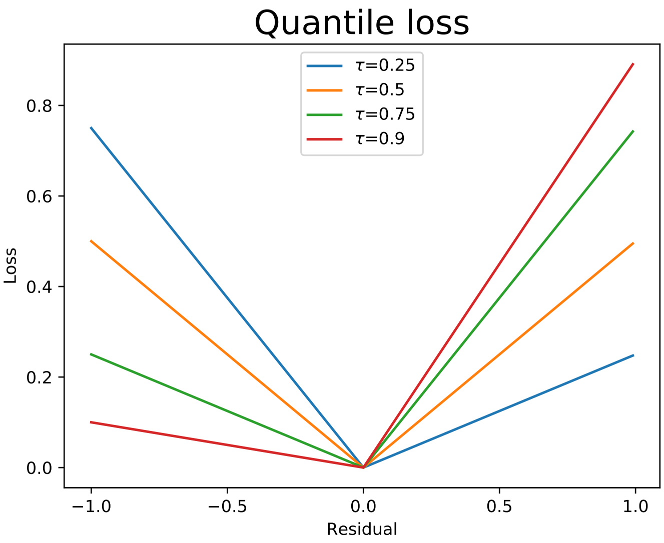

Figure A.2: Quantile loss for different quantile values

Proposition 2 in Dabney et al. (2018a) states that the combined quantile projection $\prod_{W_1}$ with the Bellman update $\mathcal{T}^\pi$ has a unique fixed point $\hat{Z}^\pi$ and the repeated application of this operator, or its stochastic approximation, converges to $\hat{Z}^\pi$.

A.1.5 Quantile Regression Temporal Difference Learning

Dabney et al. (2018a) presents a complete algorithmic approach to distributional RL consistent with previous theoretical results to approximate the value distribution with a parameterized quantile distribution over a set of quantile points and training the location parameters via quantile regression. Temporal difference learning updates the estimated value function with a single unbiased sample following policy π. Quantile regression obtains an approximation with minimal 1-Wasserstein distance from the true quantile function by observing samples $y \sim Y(x, a)$ and minimizing equation (A.13).

Quantile regression temporal difference learning algorithm combines this with the distributional Bellman operator to estimate the target distribution for quantile regression:

$$u = r + \gamma z' - \theta_i(x) \tag{A.15}$$

$$\theta_i(x) \leftarrow \theta_i(x) + \alpha(\hat{\tau}_i - \mathbb{1}_{\{u<0\}}) \tag{A.16}$$

$$a \backsim \pi(\cdot|x), r \backsim R(x,a), x' \backsim P(\cdot|x,a), z' \backsim Z_\theta(x') \tag{A.17}$$

where Z_θ is a quantile distribution as in (A.10) and $\theta_i(x)$ is the estimated value of $F^{-1}_{Z^\pi(x)}(\hat{\tau}_i)$ for state x.

A.1.6 Implicit quantile networks

In this thesis we represent the quantile distribution slightly different than (A.10). We use an implicit quantile network (IQN) instead, a deterministic parametric function trained to reparameterize samples from a base distribution e.g. $\tau \sim U([0,1])$ to the respective quantile values of a target distribution, as presented in Dabney et al. (2018b).
Hence, in this case we do not learn the distribution at uniformly separated quantile levels τ_i $(i \in \{1, N\})$ but the quantile level τ is now an input to the network instead. Despite the difference in the parameterization, the followed approach is the same as described in previous sections.
However, as discussed by the IQN authors (Dabney et al., 2018b), the contraction mapping results for fixed grid of quantiles given in Dabney et al. (2018b) is not proven yet to extend to this more general class of approximate quantile functions. Empirical performance success still motivates the use of the approach.
We also note that we slightly modify the IQN network as presented in Dabney et al. (2018b) to adapt it to the continuous setting. To this end, we add an extra input to the network, namely action a, so the output of the network is an approximation of $Z(x, a; \tau)$ for every triplet (x, a, τ), instead of outputting n action-values, one for each action a where n is the number of possible discrete actions in the environment.

Curious readers are encouraged to read the papers Bellemare et al. (2017); Dabney et al. (2018a,b) for more information about the distributional RL approach.