Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Learning &
Adaptive Systems

Master Thesis

Thesis title

Spring Term 2020

Supervised by:
Sebastian Curi

Author:
Núria Armengol Urpí

Professor:
Andreas Krause

Contents

Preface

To be written

Abstract

To be written

Chapter 1

Introduction

1.1 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. This leads to the so called *risk-neutral* behavior.

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).

Risk-sensitive decision-making provides a promising approach to compute robust and safe policies, essential in safety-critical applications, such as autonomous navigation. However, finding computationally tractable and conceptually meaningful methodologies for such a goal 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.

We can segment RL algorithms accounting for risk in two main categories: the first transform 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.
We will focus on the first category, which can be divided into 3 subcategories: worst-case criterion, constrained criterion and risk-sensitive criterion.
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 expcted 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.
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 doesn't exceed a given threshold (Tamar et al., 2012). This 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).

Our approach aims to maximize another risk metric, namely the Conditional Value at Risk (CVaR). 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 has recently gained a lot of popularity 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.

1.2 Related Work

CVaR optimization aims to find the parameters θ that maximizes $\mathrm{CVaR}_\alpha(Z)$, 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), and not much research has been done for the RL framework.

Under the RL framework, 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. (2015a) proposed a policy gradient (PG) algorithm for CVaR optimization, by deriving a sampling based estimator for the gradient of CVaR and using it to optimize the CVaR via stochastic gradient descent. 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, which has several disadvantages also shared with Tamar et al. (2015b). First, being PG algorithms, they suffer from high variance on the gradient estimates, 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 full trajectories.
Chow and Ghavamzadeh (2014) also proposed both a trajectory-based and incremental actor-critic approach which help 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.

1.3 Our approach

In this thesis, we propose a CVaR optimization approach based on a model-free, off-policy deterministic actor-critic algorithm using deep function approximators. Instead of empirically estimating the VaR from the observed rewards or using the CVaR Bellman equation, we build upon recent research in distributional RL (Bellemare et al., 2017; Dabney et al., 2018a,b) to estimate the full *value distribution* (i.e. the distribution of the random return received by a RL agent). A critic learns a parameterization of the value distribution and the actor, is trained towards policies that maximize the CVaR of the returns by sampling from the parameterized value distribution and performing stochastic gradient ascent.
In this line, we would like to mention the research by Barth-Maron et al. (2018), where they also adapt the distributional perspective on RL to the continuous setting using deterministic policy gradients. However, they use a more restrictive distribution representation and they do not focus on risk-sensitive policies but still on maximizing the expected return.

The CVaR optimization algorithm that we introduce has 2 main properties that so far, we do not have knowledge any other algorithm for CVaR optimization in the RL setting has; 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.

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.

In a similar line we would like to mention the recent research by Tang et al. (2020), in which they also use deterministic policy gradient algorithm and a distributional critic for CVaR optimization. However, their approach differs from ours since they decide to approximate the return distribution up to its 2n 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 good performance in a self-driving environment, we consider using a Gaussian approximation for the return distribution can be a bit restrictive, by the fact that it is light-tailed and unimodal. One of the motivations for using CVaR instead of other metrics is the fact that it takes into account 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 capabilities of the CVaR as a metric.

We show algorithm performance in two different settings. First, in the most common off-policy RL setting in which the environment simulator is given and the agent can interact with it to collect new data from time to time. Second, and most importantly, the agent is trained completely offline by using an external dataset and when no further interaction with the environment is allowed. For the 2 settings, the algorithm is able to learn risk-sensitive policies in complex high-dimensional, continuous action spaces such as for some modified environments from the Open AI Gym toolkit and from D4RL, a recent suite of tasks and datasets for benchmarking progress in offline RL.

final motivation sentence

We think that our algorithm ...

Chapter 2

Problem description

2.1 Problem description

Standard reinforcement learning approaches 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 Vale at Risk (CVaR).
We, thereby, present a RL algorithm that aims to find policies with optimal CVaR.

2.1.1 Reinforcement Learning

Reinforcement learning 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.

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 doesn'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, shape of the reward distribution might be unimportant, since highly different distributions still can have same expectation. However, in real world scenarios, in when 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*

As an example, suppose a participant in a game is told to choose between two doors. One door hides 1000CHF and the other 0CHF. The 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.

In this thesis we present an algorithm to find risk-sensitive policies that maximize the CVaR of the reward distribution.

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:

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

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 an algorithm for CVaR optimization using a *distributional* variant of the deterministic policy gradient algorithm. 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 *value distribution* (i.e. the distribution of the random return received by a RL agent). Having an estimation of the true value distribution, we compute the CVaR of the current policy via sampling from the estimated value distribution and we move the current policy towards maximizing it via stochastic gradient ascent.

should I repeat all this? Specify that they are NN's actor/critic?

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.

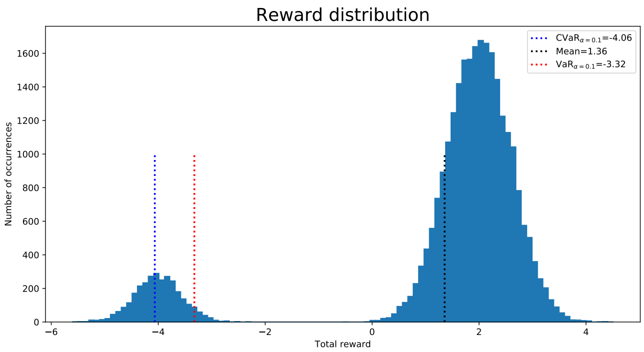

Figure 2.1: Histogram showing VaR,mean and CVaR of a sampled probability distribution. The image shows that the mean doesn't take into account low probability events and the flow of the VaR as a risk metric, since we could move the left-most mode to minus infinity and the VaR will remain the same, whereas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

Chapter 3

Distributional Deterministic Off-policy Actor-critic algorithm for CVaR optimization

We introduce an off-policy, model-free algorithm for CVaR optimization using deep function approximators that can learn policies in high-dimensional, continuous action spaces. Our work is based on the deterministic policy gradient algorithm.
Specifically, we use an actor-critic approach: the critic uses a distributional variant of RL and it is trained to estimate the whole value distribution, whereas the actor is trained via gradient ascent to maximize the CVaR of this distribution.
In the following we briefly describe the standard DPG algorithm 3.1, then we introduce our distributional approach and how we use it to obtain risk-sensitive policies 3.2 and in the last section, we focus more on implementation details of the algorithm, especially explaining more about the neural networks we use and providing a pseudocode 3.4.

Extensive and more theoretical information on the distributional RL approach is addressed in appendix A.2.

3.1 Off-policy Deterministic Actor-Critic

change title not to repeat with subsection?

3.1.1 Preliminaries

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. 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:

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

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:

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

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 $\pi^*$from it via the greedy policy $a^* = \underset{a}{\text{argmax}}Q^*(x,a)$.

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 to be able to select actions without consulting the value function.

3.1.2 Deterministic policy gradients

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 $J(\pi_\theta)$:

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

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

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 Q and policy, 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).

Unlike Q-learning, which directly attempts to learn the optimal Q-function, actor-critic methods aim to learn the Q-functions corresponding to the current parameterized policy $\pi|\theta^\pi$. In the off-policy setting, the critic parameterized by θ^Q estimates the action-value function $Q^\pi(x, a)$ *off-policy* from trajectories generated by a behavior policy β (discussed in 3.1.1) using the Bellman equation. It learns by minimizing the loss:

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

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 target 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.

Actor-critic methods are closely related with policy iteration methods (Lagoudakis and Parr, 2004). Policy iteration consists of two phases: policy evaluation and policy improvement. Policy evaluation phase computes the Q-function for the current policy π, Q^π, by solving for the fixed point such that $Q^\pi = \mathcal{T}^\pi Q^\pi$, which can be done via gradient updates of the loss 3.7. Policy improvement is done by choosing the action that greedily maximizes the Q-value at each state which can be done by using a gradient based update procedure as in 3.8.

3.2 The algorithm: Distributional off-policy deterministic actor critic

We present the CVaR optimization algorithm which is the main contribution 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 $F(z) := P(Z \le z)$ 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 critic IQN $Z(x,a;\tau|\theta^Z)$ 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 critic IQN using the sampled quantile regression loss (Koenker, 2005) on the pairwise temporal-difference (TD)-errors. For two samples $\tau, \tau' \sim U([0,1])$, and current policy π_{θ^π}, the sampled TD error is:

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

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 assymetric squared loss in an interval $[-\kappa, \kappa]$ around zero and reverts to a standard quantile loss outside this interval.
The Huber loss is given by Huber (1964):

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

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

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

Quantile loss penalizes overestimation errors (u<0) with weight $1-\tau$ and underestimation errors (u>0)with weight τ. By minimizing 3.10 via stochastic gradient descent with respect to θ^Z we aim to move towards the true quantile function.

Mentioned in Dabney et al. (2018b) can the contraction mapping results for a fixed grid of quantiles given by Dabney et al. (2018) be extended to the more general class of approximate quantile functions studied in this work?

3.2.2 Actor

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

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

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

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 actually 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 as mapping $\phi(\tau) = \alpha\tau$, 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 *deterministic risk-sensitive policy gradient*, for the parameters θ^π of the actor:

call it like that?

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

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

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

3.3 Summary

The algorithm can be summed up with:

1. Update distributional critic IQN via off-policy quantile-regression TD-learning:
$$\delta^{\tau,\tau'} = r + \gamma Z(x_{t+1}, \pi(x_{t+1}|\theta^{\pi}); \tau'|\theta^{Z}) - Z(x_t, a_t; \tau|\theta^{Z}) \tag{3.22}$$
2. Update actor via deterministic risk-sensitive policy gradient ascent:
$$\nabla_{\theta^{\pi}} J_{\beta}(\pi|\theta^{\pi}) \approx \mathbb{E}_{x\sim\rho^{\beta}}\left[\nabla_{\theta^{\pi}}\pi(x,|\theta^{\pi})\nabla_{a}[\frac{1}{\alpha}\frac{1}{K}\sum_{i=1}^{K} Z(x,a;\tau_i)|\theta^{Z})]|_{a=\pi(x|\theta^{\pi})}\right] \tag{3.23}$$

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 due to the deterministic essence of the policies, which removes the integral over actions in the policy gradient and the expected value over actions in the Bellman equation for the critic.

3.4 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). 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.

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

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).

explain more?

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* and *actor target network* ($\hat{Z}(x, a; \tau|\theta^{\hat{Z}})$ and $\hat{\pi}(x|\theta^{\hat{\pi}})$) 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 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 networks updates for every two critic network updates.

Second, we use target policy smoothing to address a particular failure mode that can happen in DeepDPG. 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:

$$a_{t+1} = \hat{\pi}(x_{t+1}|\theta^{\hat{\pi}} + \text{clip}(\epsilon, -c, c) \quad \epsilon \sim \mathcal{N}(0, \sigma) \tag{3.25}$$

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

To compute the quantile regression loss, we used N'=N=32 quantile levels to sample from target 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 $Z(x, a; \tau)|\theta^{Z})$, we use K=8 quantile levels.

We start training after a warm-up of T=25000 environment interactions. During this time, in order to enhance exploration, we don't use actor networks but random policies instead. We use Adam optimizer for learning the neural networks parameters with learning rates of 10^{-4} and 10^{-3} for the actor and the critic respectively. For additional information on the networks used, please see Appendix A

Algorithm 1: How to write algorithms

```
Result: Write here the result
initialization;
while While condition do
    instructions;
    if condition then
        instructions1;
        instructions2;
    else
        instructions3;
    end
end
```

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.

use replay buffer, target networks as DQN to deal with problems of Q-learning with functions approximators.

Since distorted expectations can be expressed as weighted average over the quantiles Dhaene et al. (2012), we can use a specific sampling base distribution $\beta : [0, 1] \rightarrow [0, 1]$ to sample the quantile levels $\tau \in [0, 1])$ from our critic network $Z(x, a; \tau)$.

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., 2015a) don't show such property.

For the CVaR we have the distribution:

$$\mathrm{CVaR}(\alpha, \tau) = \alpha\tau \quad \tau \sim U([0, 1]) \tag{3.26}$$

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

$$\mathrm{Wang}(\eta, \tau) = \Phi(\Phi^{-1}(\tau) + \eta) \quad \tau \sim U([0, 1]) \tag{3.27}$$

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:

$$\mathrm{Pow}(\eta, \tau) = \begin{cases} \tau^{\frac{1}{1+|\eta|}} & \text{if } \eta \geq 0 \\ 1 - (1 - \tau)^{\frac{1}{1+|\eta|}} & \text{otherwise} \end{cases} \qquad \tau \sim U([0, 1]) \tag{3.28}$$

3.6 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:

$$\mathrm{CVaR}_\alpha(Z) = \max_\nu \left\{ \nu + \frac{1}{\alpha} \mathbb{E}_Z[[Z - \nu]^-] \right\} \tag{3.29}$$

which in the optimal point it holds that $\nu^* = \mathrm{VaR}_\alpha(Z)$.
Given the fact that our critic network estimates the $\mathrm{VaR}_\alpha(Z)$ of the return distribution, we can compute the CVaR using 3.29 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:

$$\mathrm{CVaR}_\alpha(Z) \approx \nu + \frac{1}{\alpha}\frac{1}{K}\sum_{i=1}^{K}\left[Z(x,a;\tau_i|\theta^Z) - \nu\right]^- \tag{3.30}$$

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). For the sake of comparison we present it in the next section.

Chapter 4

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 $\mathcal{D}$ and each new policy π_k collects additional data, such that $\mathcal{D}$ 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 $\mathcal{D}$ 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. Off-line RL holds tremendous promise for making it 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).

4.1 Issues with Batch RL

change title?

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 $d^{\pi_\beta}(s)$. 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 off-line 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 (as ours, i.e. methods aiming to learn a state or state-action value function), 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 d^{π_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.

maybe cite more?

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

4.2 Our approach

more explicit title?

In Fujimoto et al. (2019), they present a Batch-Constrained Deep Q-learning (BCQ) algorithm in which they train a generative model G_w to generate actions with high similarity to the dataset. For the generative model they use a conditional variational auto-encoder (VAE) Kingma and Welling (2014) which generates action samples as a reasonable approximation to $\underset{a}{\text{argmax}} P^G_{\mathcal{B}}(a|x)$, where $P^G_{\mathcal{B}}(a|x)$ is the state-conditioned marginal likelihood given the (state-action) pairs in the batch $\mathcal{B}$. Fujimoto et al. (2019) presents a Q-learning approach, for which they sample multiple candidate actions from the VAE, they perturbed them via a perturbation model and select the highest valued-action according to a learnt Q-network.

For our approach, we only use from Fujimoto et al. (2018) the VAE network to sample actions similar to the dataset, which are then perturbed to maximize the CVaR of the return distribution. The rest of the algorithm keeps intact as the one we presented, and as later summarized.

We proceed to explain how we update our Distributional off-policy deterministic actor-critic algorithm previously presented to be able to learn effectively in the Batch RL setting.

We generate actions using a variational autoencoder G_w, which is trained 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. This perturbation model is trained the same way as the previously presented actor network, i.e. to maximize the CVaR of the estimated return distribution via deterministic gradient ascent.

$$\hat{a}_i \sim G_w(x) \tag{4.1}$$

$$a_i = \hat{a}_i + \xi(x, \hat{a}_i; \phi, \theta^{\xi}) \tag{4.2}$$

where $\xi(s, a; \phi)$ is a perturbation 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 will reduce the perturbation and hence the learnt policy will behave more closely to the behavior policy used to obtain the dataset.

4.2.1 Details of Conditional VAE

For the generative model G_w, we use a conditional variational autoencoder G_w. Given state-action pairs x^B, a^B from the dataset, the conditional variational autoencoder aims to maximize the state-conditioned marginal log-likelihood $\log P_{\mathcal{B}}(A|X) = \sum_{i=1}^{N} \log P_{\mathcal{B}}(a_i|x_i)$. Hence, using the trained generative model we can sample actions as a reasonable approximation to $\arg\max_a P^G_{\mathcal{B}}(a|x)$, ie very similar to the actions from the dataset.

It is assumed that the data is generated by some random process, involving an unobserved continuous random variable **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 $P^G_{\mathcal{B}}(a|x)$. 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:

$$\log p(A|X;\theta) \geq \mathcal{L}(\theta,\phi;A|X) \tag{4.3}$$

$$\mathcal{L}(\theta,\phi;A|X) = \mathbb{E}_{q(z|A,X;\phi)}[\log p(A|z,X;\theta)] - D_{KL}(q(z|A,X;\phi) \mid\mid p(z;\theta)) \tag{4.4}$$

4.3 Batch RL Distributional DeterministicOff-policy Actor-critic algoritm for CVaR otpimization

For our implementation, the prior $p(z;\theta)$ 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 and decoders we use neural networks. For the encoder $q(z|a,x;\phi)$ we used a neural network with Gaussian output, specifically a multivariate Gaussian with a diagonal covariance structure $\mathcal{N}(z|\mu(a,x;\phi),\sigma^2(a,x)Id;\phi)$, where μ and σ are the outputs of the neural network, i.e. nonlinear functions of datapoint (x_i,a_i) and 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 another neural network with deterministic output, i.e. nonlinear function of datapoint (x_i,z_i) and 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 4.4 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.

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.

The perturbation model is trained as in 3.23:

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

4.4 Technical details

Algorithm works very similarly to the off-policy RL previously presented, and hence shares almost all details as the ones introduced in 3.4. We focus here on the differences or new additions. For batch RL, we do not need to account for exploration so there is no such behavior policy. At every training-step a minibatch of observations $(x_t^B, a_t^B, r_t^B, x_{t+1}^B)$ is sampled from static dataset.

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 we sample actions $hata_k^{VAE}$ 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:

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

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

Actor or *perturbation model* training is done by gradient ascent as in 4.7 where

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

We train the critic in the same way as the original algorithm, using a critic target network to estimate the target values which slowly tracks the learnt networks. a_{k+1} actions used to compute target Z values $Z(x_{k+1}^B, a_{k+1}; \tau_i)$, are computed as the output of the perturbation model ϵ, which also uses a target network.

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 $\sigma = 0$.

Chapter 5

Results

In this section we show the performance of our algorithm across a variety of continuous control tasks. First, we show its 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 its performance on several external datasets provided by the D4RL suite. 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 our algorithm in an off-policy setting. We show that when the reward is deterministic, the policies learnt are the same. When adding uncertainty, whe 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 $\Delta k = 0.1$ 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:

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

$$x_{k+1} = x_k + v_k\Delta k + 0.5a_k\big(\Delta k\big)^2 \tag{5.2}$$

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 with 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* learns 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

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 with 24 time-steps. Hence the final cumulative reward $G_T = (24-1)R_k + R_F = 140$.

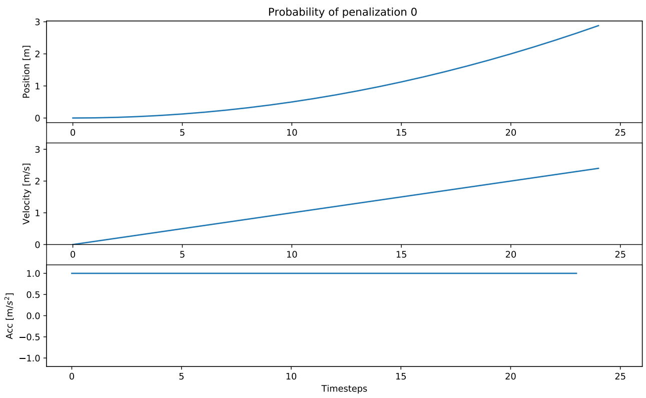

Figure 5.1: States and control evolution during evaluation without velocity penalization. Learnt policy is the same for both *Mean* and *CVaR*

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. Hence the final cumulative reward $G_T = (10+21-1)R_k + R_F = 70$.

We note that the reward values were chosen in order to make sure that, for the Setting 2, driving with a velocity higher than $1\mathrm{m\,ts^{-1}}$ never induces higher cumu-

lative rewards. In that case, it will reach goal with 24 time-steps but would add penalization for driving for 10 time-steps in a penalized velocity:
$G_T = (24-1)R_k + (21-1)R_v + R_F << (10+21-1)R_k + R_F = 70$

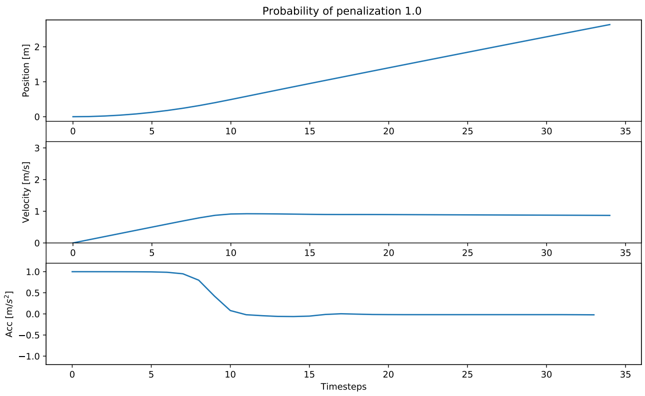

Figure 5.2: States and control evolution during evaluation with velocity penalization with probability 1. Learnt policy is the same for both *Mean* and *CVaR*

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 5 training episodes, we evaluate the current policy for 500 evaluation episodes. During evaluation, we remove the exploration noise from the policy. During this 500 evaluation episodes, we compute the mean of the cumulative rewards, the CVaR with a confidence level of 0.1 and also the mean of the number of times the maximum velocity is exceeded.

After training, we used the final policies and deployed them in the environment. Final trajectories are shown below. An histogram of the resulting cumulative rewards is added below showing how the *Mean* algorithm obtain 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*. Results are shown below.

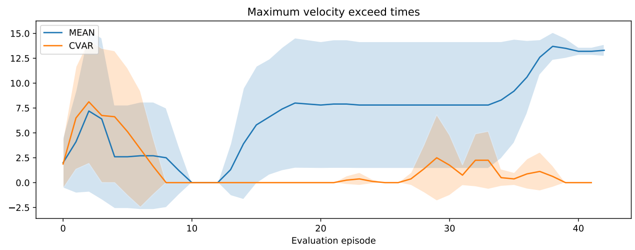

Figure 5.3: Evolution of number of times penalized velocity is exceeded. Every datapoint corresponds to one evaluation process performed every 10 training episodes. For the plot we show averaged values over 5 random seeds and 1 std

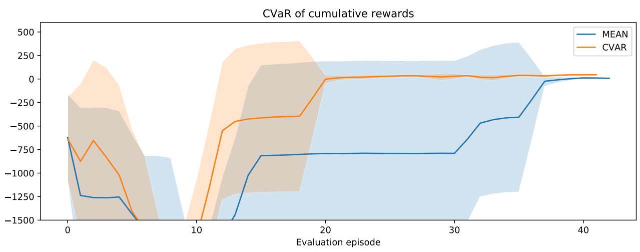

Figure 5.4: Evolution of $\mathrm{CVaR}_\alpha = 0.1$ of the cumulative rewards over 500 evaluation episodes. Every datapoint corresponds to one evaluation process performed every 10 training episodes. For the plot we show averaged values over 5 random seeds and 1 std

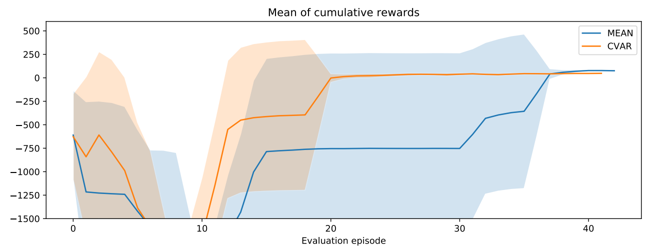

Figure 5.5: Evolution of mean of the cumulative rewards over 500 evaluation episodes. Every datapoint corresponds to one evaluation process performed every 10 training episodes.For the plot we show averaged values over 5 random seeds and 1 std

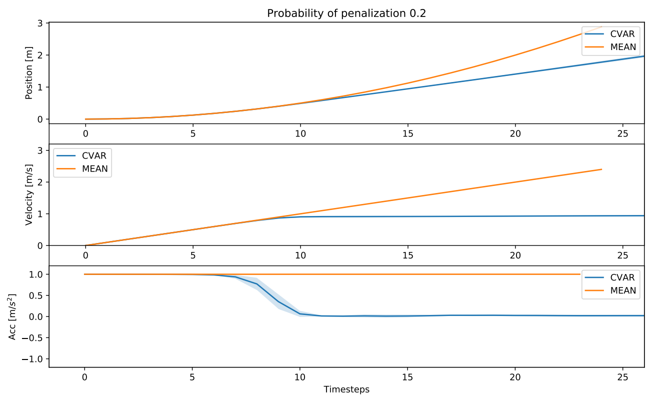

Figure 5.6: States and control evolution during evaluation with velocity penalization with probability 0.2. Learnt policies by *Mean* and *CVaR* differ, *CVaR* leading towards risk-averse behavior

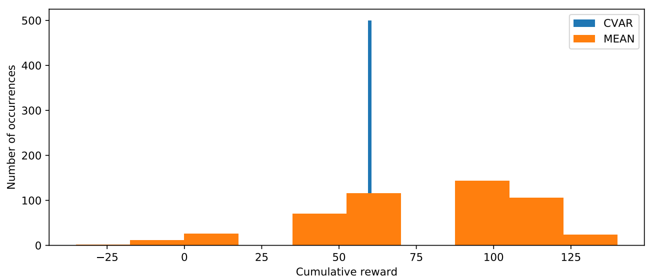

Figure 5.7: Comparison of cumulative rewards achieved with *CVaR* and *Mean* when the probability of velocity penalization is 0.2. *Mean* achieves a higher expected value ($\mu = 78.1$) but has a lower CVAR ($\text{CVaR}_{\alpha=0.1}$=12) compared to *Mean* ($\mu = 60$ and $\text{CVaR}_{\alpha=0.1}$=60). Data was obtained by acting for 500 episodes with the trained policies. Distribution of rewards for *Mean* follows a binomial distribution B(n,p) with n being the number of steps with a velocity higher than the penalized and p the probability of penalization.

5.2 Batch RL Setting

We test the performance of our algorithm in a Batch RL setting. To this end, we make use the D4RL dataset Fu et al. (2020), a recent suite of tasks and datasets for benchmarking progress in offline RL.

5.2.1 Half-Cheetah

We first use the Gym-Mujoco benchmark tasks, specifically *HalfCheetah*. In the *HalfCheetah* environment, a 2D chetah 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.

Dataset

We use the *halfcheetah-medium-v0* dataset, which consists of data generated by training a policy online using a standard RL algorithm (SAC) for a few steps till it reaches approximately 1/3 the performance of the expert. Then, they stop the training and collecting 1M samples from this partially trained policy.

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. A reward of $R_v = -100$ is given with probability 0.05, if the velocity of the cheetah is greater than 4. 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 open-loop, to collect the desired information and modify the rewards accordingly. The modified dataset was used to train the algorithm in an off-line way.

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. Only for this evaluation process, we allow the agent to interact with the environment. The environment is modified with a RewardWrapper to introduce the stochasticity on the reward for high velocities. During evaluation, to ensure the deterministic nature of the policy, the latent vector is not sampled from a gaussian, but a vector of zeros is used instead.

During this 5 evaluation episodes, we compute the mean of the cumulative rewards, the CVaR with a confidence level of 0.1 and also the mean of the number of times the maximum velocity is exceeded.

After training, we used the final policies and deployed them in the noisy environment for 20 episodes 200 timesteps each. An histogram of the resulting cumulative rewards is added below showing how the *Mean* algorithm obtain 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*. Results are shown below.

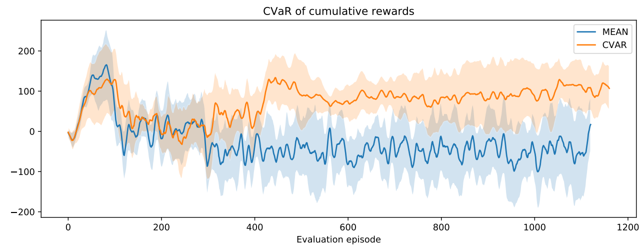

Figure 5.8: $\text{CVaR}_{\alpha} = 0.1$ of the cumulative rewards over 5 evaluation episodes. Every point corresponds to one evaluation process performed every 1000 steps of training.For the plot we show averaged values over 5 random seeds and 1 std

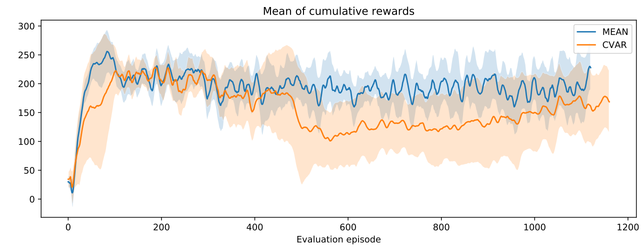

Figure 5.9: Mean of the cumulative rewards over 5 evaluation episodes. Every point corresponds to one evaluation process performed every 1000 steps of training.For the plot we show averaged values over 5 random seeds and 1 std

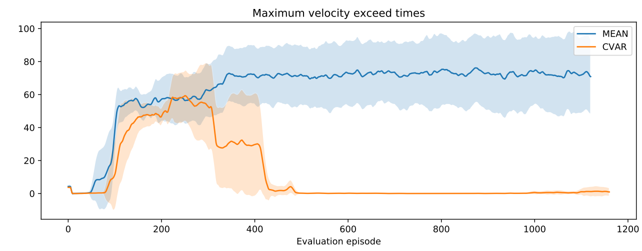

Figure 5.10: Mean of number of times the maximum velocity was exceeded over 5 evaluation episodes. Every point corresponds to one evaluation process performed every 1000 steps of training. For the plot we show averaged values over 5 random seeds and 1 std

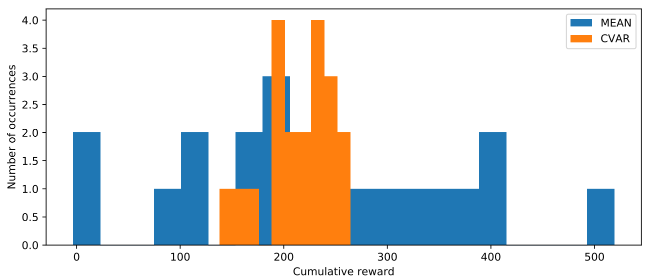

Figure 5.11: Histogram of cumulative rewards during 200 time steps using the trained final policies

5.2.2 Walker

Add more environments. To be decided

5.2.3 Medical environment

Add more environments. To be decided

Chapter 6

Discussion

To be written.

Bibliography

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

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.

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.

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.

A. Tamar, Y. Chow, M. Ghavamzadeh, and S. Mannor, "Policy gradient for coherent risk measures," *Advances in Neural Information Processing Systems*, vol. 2015-Janua, pp. 1468–1476, 2015.

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.

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.

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

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.

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.

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.

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

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.

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

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.

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.

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.

J. Dhaene, A. Kukush, D. Linders, and Q. Tang, "Remarks on quantiles and distortion risk measures," *European Actuarial Journal*, 2012.

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.

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

Appendix A

A.1 Experiment details

Extra info about the network architectures?

!

A.2 Distributional RL

demonstrated the importance of learning the value distribution, i.e., the distribution of the random return received by a RL agent. This differs from the common RL approach which is focused on learning the expected value of this return.

To be rewritten

cite papers

Goal in RL is to teach an agent to act so that it maximizes its expected utility, Q Sutton and Barto (1998) Bellman's equation describes this value Q in terms of the expected reward and expected outcome of the random transition $(x, a) \to (X', A')$, showing the particular recursive relationship between the value of a state and the values of its successor states:

$$Q(x, a) = \mathbb{E}[R(x, a)] + \gamma \mathbb{E}[Q(X', A')] \tag{A.1}$$

Distributional RL aims to go beyond the notion of *value* and training to study instead the random return Z.

A.2.1 Example showing interest in learning the distribution

Imagine the example in which we are playing a board game and we roll 2 dices. If we get a 3, we fall in prison and need to pay 2000CHF (i.e. reward of -2000CHF), whereas otherwise we we collect a salary of 200CHF (i.e. reward of +200CHF). If we consider the common reinforcement learning approach and we compute the expected immediate ($\gamma = 1$) reward:

remove and maybe the motivation is done for the doors example in chapter 2

$$\mathbb{E}[R(x)] = \frac{1}{36}(-2000\,\text{CHF}) + \frac{35}{36}(200\,\text{CHF}) = 138.88\,\text{CHF} \tag{A.2}$$

Hence, the expected immediate return is +138.88CHF. However, in any case we will get a return of +138.88CHF. Instead:

$$R(x) = \begin{cases} -2000\,\text{CHF}, & \text{w.p } \frac{1}{36} \\ 200\,\text{CHF}, & \text{w.p } \frac{35}{36} \end{cases}$$

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.

add new section

This variable captures intrinsic randomness from:

1. Immediate rewards
2. Stochastic dynamics
3. Possibly an stochastic policy

Having defined $Z^{\pi}(x,a)$, we can clearly see that:

$$Q^{\pi}(x,a) = \mathbb{E}[Z^{\pi}(x,a)] \tag{A.3}$$

Z is also described by a recursive equation, but of a distributional nature:

$$Z^{\pi}(x,a) \overset{D}{=} R(x,a) + \gamma Z(x',a') \tag{A.4}$$

where $x' \backsim p(\cdot|x,a)$ and $a' \backsim \pi(\cdot|x')$

where $\overset{D}{=}$ denotes that the RV on both sides of the equation share the same probability distribution. The *distributional Bellman equation* defined in (A.4), states that the distribution of Z is characterized by the interaction of 3 RV's: the random variable 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 *value distribution.*

A.2.2 Distributional Bellman Operator

In the policy evaluation setting Sutton and Barto (1998), one aims to find the value function $V\pi$ associated with a given fixed policy π. In the distributional case, we aim to find $Z\pi$. Bellemare et al. (2017) defined the Distributional Bellman operator T^{π}. We view the reward function as a random vector R $\in \mathbb{Z}$ and define the transition operator $P^{\pi} : \mathbb{Z} \to \mathbb{Z}$

$$P^{\pi} Z(x,a) \overset{D}{=} Z(X',A') \tag{A.5}$$
$$X' \backsim P(\cdot|x,a) \text{ and } A' \backsim \pi(\cdot|X') \tag{A.6}$$

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

$$T^{\pi} Z(x,a) \overset{D}{=} R(x,a) + \gamma P^{\pi} Z(x,a) \tag{A.7}$$

Bellemare et al. (2017) showed that (A.7) 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 (CDF). Tht is, the p-Wasserstein metric between distributions U and Y is given by:

$$W_p(U, Y) = \Big(\int_0^1 |F_Y^{-1}(w) - F_U^{-1}(w)|^p dw \Big)^{\frac{1}{p}} \tag{A.8}$$

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

$$F_Y^{-1}(w) := \inf\big\{y \in \mathbb{R} \,|w \ \leq F_Y(w)\big\} \tag{A.9}$$

where $F_Y(w) = Pr(y \leq Y)$.

Add Figure 2.1 in Dabney et al. (2018a)

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 $\hat{d}_p$:

Let $\mathcal{Z}$ be the space of action-value distributions:

$$\mathcal{Z} = \big\{Z \mid \mathcal{X} \times \mathcal{A} \rightarrow \wp(\mathbb{R}) \tag{A.10}$$

$$\mathbb{E}[|Z(x, a)|^p < \infty, \forall(x, a), p \geq 1]\big\} \tag{A.11}$$

check first line the $\wp$

Then, for two action-value distribution $Z_1, Z_2 \in \mathcal{Z}$, the maximal form of the Wasserstein metric is defined by:

$$\hat{d}_p(Z_1, Z_2) := \sup_{x,a} W_p(Z_1(x, a), Z_2(x, a)) \tag{A.12}$$

Bellemare et al. (2017) showed that $\hat{d}_p$ is a metric over value distributions and furthermore, the distributional Bellman operator T^π is a contraction in $\hat{d}_p$. Consider the process $Z_{k+1} := T^\pi Z_k$, starting with some $Z_0 \in \mathcal{Z}$.

$T^\pi Z : \mathcal{Z} \rightarrow \mathcal{Z}$ is a γ-contraction in the Wasserstein metric $\hat{d}_p$, which implies that not only the first moment (expectation) converges exponentially to Q^π, but also in all moments.

Lemma 1: (Lemma 3 in Bellemare et al. (2017))

T^π is a γ-contraction: for any two $Z_1, Z_2 \in \mathcal{Z}$,

$$\hat{d}_p(T^\pi Z_1, T^\pi Z_2) \leq \gamma \hat{d}_p(Z_1, Z_2) \tag{A.13}$$

Using Banach's fixed point theorem, it is proven that T^π has a unique fixed point, which by inspection must be Z^π.

Hence the $\hat{d}_p$ metric is shown to be useful metric for studying behavior of distributional RL algorithms, and to showed their convergence to a fixed point. Moreover, shows than en effective way to learn a value distribution is to attempt minimize 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.

We have so fare considered a policy evaluation setting, i.e. trying to learn a value distribution for a fixed policy π, and we studied the behavior of its associated distributional operator T^{π}. In the control setting, i.e., when we try to find a policy π^* that maximizes a value, or its distributional analogous, i.e. that induces an optimal value distribution. However, while all optimal policies attain the same value Q^*, in general there are many optimal value distributions.

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.*.
Let Π^* be the set of optimal policies.
Definition 1: An optimal value distribution is the value distribution of an optimal policy. The set of optimal value distributions is

$$\mathcal{Z}^* := \left\{ Z^{\pi^*} | \pi^* \in \Pi^* \right\}$$

Not all value distributions with expectation Q^* are optimal, but they must match the full distribution of the return under some optimal policy. **Definition 2:** A greedy policy π for Z $in \mathcal{Z}$ maximizes the expectation of Z. The set of greedy policies for Z is:

$$\mathcal{G}_{\mathcal{Z}} := \left\{ \pi \mid \sum_{a} \pi(a|x) \mathbb{E}(Z(x,a) = \max_{a' \in \mathcal{A}} Q(x',a') \right\}$$

We will call a *distributional Bellman optimality operator* any operator $\mathcal{T}$ which implements a greedy selection rule, i.e.:

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

As in the policy evaluation setting, we are interested in the behavior of the iterates $Z_{k+1} := \mathcal{T}Z_k, Z_0 \in \mathcal{Z}$. Lemma 4 in Bellemare et al. (2017) shows that $\mathbb{E}Z_k$ behaves as expected: **Lemma 4:** Let $Z_1, Z_2 \in \mathcal{Z}$. Then:

$$||\mathbb{E}\mathcal{T}Z_1 - \mathbb{E}\mathcal{T}Z_2\|_{\infty} \leq \gamma ||\mathbb{E}Z_1 - \mathbb{E}Z_2\|_{\infty}$$

and in particular $\mathbb{E}Z_1 \to Q^*$ exponentially quickly. However, Z_k is not assured to converge to a fixed point. Specifically, they provide a number of negative results concerning $\mathcal{T}$:

Proposition 1: The operator $\mathcal{T}$ is not a contraction.
Proposition 2: Not all optimality operators have a fixed point $Z^* = \mathcal{T}Z^*$ **Proposition 3**: That $\mathcal{T}$ has a fixed point $Z^* = \mathcal{T}Z^*$ is insufficient to guarantee the convergence of $\{Z_k\}$ to Z^*

Another result, 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.2.3 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.

A.2.4 Quantile projection:

Our current aim is to estimate quantiles of the target distribution, i.e. the values of the return that divide the value distribution in equally sized parts. We will call it a quantile distribution, and we will let $\mathcal{Z}_Q$ be the space of quantile distributions. We denote the cumulative probabilities associated with such a distribution 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:

$$Z_\theta(x,a) \coloneqq \frac{1}{N}\sum_{i=1}^{N}\delta_{\theta_i(x,a)} \tag{A.14}$$

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

$$\prod{}_{W_1} Z \coloneqq \underset{Z_\theta \in \mathcal{Z}_Q}{\arg\min} W_1(Z, Z_\theta) \tag{A.15}$$

This projection $\prod_{W_1}$ is the quantile projection.
We can quantify the projection between a distribution with bounded first moment Y and U, a uniform distribution over N Diracs as in (A.14) with support $\{\theta_1, ...\theta_N\}$ by:

$$W_1(Y,U) = \sum_{i=1}^{N}\int_{\tau_{i-1}}^{\tau_i} |F_Y^{-1}(w) - \theta_i| dw \tag{A.16}$$

Lemma 2 in Dabney et al. (2018a) establishes that the values $\{\theta_1, ..., \theta_N\}$ for the returns that minimize $W_1(Y,U)$ are given by $\theta_i = F_Y^{-1}(\hat{\tau}_i)$, where $\hat{\tau}_i = \frac{\tau_{i-1}+\tau_i}{2}$.

A.2.5 Quantile Regression

Quantile regression is a method for approximating quantile functions of a distribution at specific points, i.e. its inverse cumulative distribution function. 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:

$$\mathcal{L}_{QR}^{\tau}(\theta) = \mathbb{E}_{\hat{Z}\backsim Z}[\rho_\tau(\hat{Z}-\theta)] \tag{A.17}$$
$$\rho_\tau(u) = u(\tau - \delta_{u<0}), \forall u \in \mathbb{R}$$

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), which 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 can claim that the values of $\{\theta_1, ...\theta_N\}$ are the minimizers of the following objective:

$$\sum_{i=1}^{N} \mathbb{E}_{\hat{Z}\sim Z}[\rho_{\hat{\tau}_i}(\hat{Z}-\theta_i)] \tag{A.18}$$

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

add huberloss

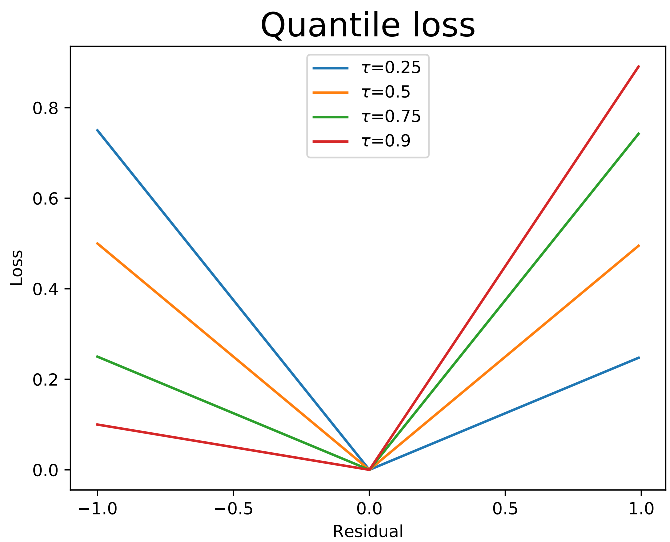

Figure A.1: Quantile loss for different quantile values

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.2.6 Quantile Regression Temporal Difference Learning

Temporal difference learning updates the estimated value function with a single unbiased sample following policy π. Quantile regression allows to improve the estimate of the quantile function for some target distribution Y(x), by observing samples $y \sim Y(x)$ and minimizing equation (A.17). Using the quantile regression loss, we can obtain an approximation with minimal 1-Wasserstein distance from the original. We can combine this with the distributional Bellman operator to give a target distribution for quantile regression, creating the quantile regression temporal difference learning algorithm:

$$u = r + \gamma z' - \theta_i(x) \tag{A.19}$$

$$\theta_i(x) \leftarrow \theta_i(x) + \alpha(\hat{\tau}_i - \delta_{u<0}) \tag{A.20}$$

$$a \backsim \pi(\cdot|x), r \backsim R(x,a), x' \backsim P(\cdot|x,a), z' \backsim Z_\theta(x') \tag{A.21}$$

where Z_θ is a quantile distribution as in (A.14) and $\theta_i(x)$ is the estimated value of $F^{-1}_{Z^\pi(x)}(\hat{\tau}_i)$ in state x.