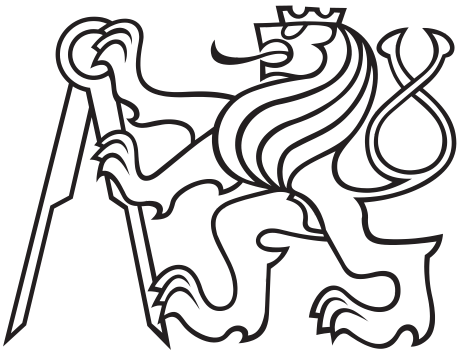
RISK-AVERSE DISTRIBUTIONAL REINFORCEMENT LEARNING

A CVAR OPTIMIZATION APPROACH

SILVESTR STANKO



Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University

May, 2018

ABSTRACT

Conditional Value-at-Risk (CVaR) is a well-known measure of risk that has been used for decades in the financial sector and has been directly equated to robustness, an important component of Artificial Intelligence (AI) safety. In this thesis we focus on optimizing CVaR in the context of Reinforcement Learning, a branch of Machine Learning that has brought significant attention to AI due to its generality and potential.

As a first original contribution, we extend the CVaR Value Iteration algorithm (Chow et al. [20]) by utilizing the distributional nature of the CVaR objective. The proposed extension reduces computational complexity of the original algorithm from polynomial to linear and we prove it is equivalent to the said algorithm for continuous distributions.

Secondly, based on the improved procedure, we propose a sampling version of CVaR Value Iteration we call CVaR Q-learning. We also derive a distributional policy improvement algorithm, prove its validity, and later use it as a heuristic for extracting the optimal policy from the converged CVaR Q-learning algorithm.

Finally, to show the scalability of our method, we propose an approximate Q-learning algorithm by reformulating the CVaR Temporal Difference update rule as a loss function which we later use in a deep learning context.

All proposed methods are experimentally analyzed, using a risk-sensitive gridworld environment for CVaR Value Iteration and Q-learning and a challenging visual environment for the approximate CVaR Q-learning algorithm. All trained agents are able to learn risk-sensitive policies, including the Deep CVaR Q-learning agent which learns how to avoid risk from raw pixels.

ACKNOWLEDGMENTS

Put your acknowledgments here.

Many thanks to everybody who already sent me a postcard!

Regarding the typography and other help, many thanks go to Marco Kuhlmann, Philipp Lehman, Lothar Schlesier, Jim Young, Lorenzo Pantieri and Enrico Gregorio[1], Jörg Sommer, Joachim Köstler, Daniel Gottschlag, Denis Aydin, Paride Legovini, Steffen Prochnow, Nicolas Repp, Hinrich Harms, Roland Winkler, Jörg Weber, Henri Menke, Claus Lahiri, Clemens Niederberger, Stefano Bragaglia, Jörn Hees, Scott Lowe, Dave Howcroft, and the whole LaTeX-community for support, ideas and some great software.

Regarding LyX: The LyX port was intially done by *Nicholas Mariette* in March 2009 and continued by *Ivo Pletikosić* in 2011. Thank you very much for your work and for the contributions to the original style.

1 Members of GuIT (Gruppo Italiano Utilizzatori di TeX e LaTeX)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

1

INTRODUCTION

A staple of an intelligent agent is the ability to reason and act over time in an environment, while working towards a desirable goal. This is the setting explored in reinforcement learning (RL), a branch of machine learning that focuses on dynamic decision making in unknown environments.

Recent advances in artificial intelligence (AI) are encouraging governments and corporations to deploy AI in high-stakes settings including driving cars autonomously, managing the power grid, trading on stock exchanges, and controlling autonomous weapon systems. As the industry steps away from specialized AI systems towards more general solutions, the demand for safe approaches to artificial intelligence increases.

In this thesis, we tackle one aspect of safe reinforcement learning, robustness, by considering the risk involved in acting in a non-deterministic, noisy environment.

1.1 MOTIVATION

Lately, there has been a surge of successes in machine learning research and applications, ranging from visual object detection [35] to machine translation [5]. Reinforcement learning has also been a part of this success, with excelent results regarding human-level control in computer games [40] or beating the best human players in the game of Go [51]. While these successes are certainly respectable and of great importance, reinforcement learning still has a long way to go before being applied on critical real-world decision-making tasks. This is partially caused by concerns of safety, as mistakes can be costly in the real world.

One of the problems encountered when training a reinforcement learning agent is sample efficiency, or the large amount of training time needed for the agent to successfully learn new and correct behaviors. The solution used by many is to train the agent in simulation - it is indeed faster (as the simulation can run in parallel or faster than real-time), safer (we do not face any real danger in simulations) and cheaper than to train the agent in the real world. This approach then raises the question whether an agent trained in simulation would perform well outside of the simulation.

Robustness, or distributional shift, is one of the identified issues of AI safety [2, 37] directly tied to the discrepancies between the environment the agent trains on and is tested on. Chow et al. [20] have shown that risk, a measure of uncertainty of the potential loss/reward, can be seen as equal to robustness, taking into account the differences during train- and test-time. This point is discussed in more detail in chapter 2 and we use it as further motivation for pursuing risk-averse objectives.

While the term risk is a general one, we will focus on a concrete notion of risk - a particular risk metric called Conditional Value-at-Risk (CVaR). Due to its favorable computational properties, CVaR has been recognized as the industry standard for measuring risk in finance, as in 2014 the Basel Committee on Banking Supervision changed its guidelines for banks to replace VaR (a previously used metric) with CVaR for assessing market risk [21]. The metric also satisfies the recently proposed axioms of risk in robotics [38].

Aside from robustness, another motivational point might be one of general decision-making. Commonly encountered in finance, decision makers face the problem of maximizing profits while keeping the risks to a minimum. The solutions to problems encountered in this thesis can therefore be seen as ones of general time-dependent risk-averse decision making.

The aim of this thesis is to consider reinforcement learning agents that maximize Conditional Value-at-Risk instead of the usual expected value, hereby learning a robust, risk-averse policy. The word *distributional* in the title emphasizes that our approach takes inspirations from the recent advances in distributional reinforcement learning [10, 23].

1.2 THESIS OUTLINE AND ORIGINAL CONTRIBUTIONS

We begin the thesis with a preliminary Chapter 2 that focuses on introducing all necessary concepts to understand the rest of this thesis. We start by formally defining the reinforcement learning framework we work with and familiarize the reader with distributional reinforcement learning. This is followed by a brief introduction to risk and risk-sensitivity, together with a formal definition of Conditional Value-at-Risk and the exact problem tackled in this thesis in Section 2.4. We end the chapter with a short literature survey.

In Chapter 3 we remind the reader of the standard Value Iteration algorithm and describe the CVaR Value Iteration algorithm (Chow et al. [20]) in detail, together with its practical discretized variant. We follow up with the first original contribution of this thesis.

1. **Fast CVaR Value Iteration:** We leverage a connection between the αCVaR_α function and the quantile function of the underlying distributions, and propose a procedure for computing fast CVaR Value Iteration updates. The original approach requires computing a linear program each iteration, separately for each probability atom. In contrast, our proposed procedure is linear in time and therefore allows running CVaR Value Iteration on much larger environments.

   We demonstrate the validity of the improved procedure by showing it is equivalent to solving the original convex problem. We formally prove this for strictly increasing distributions and empirically verify the algorithm with general distributions.

The true strength of reinforcement learning lies in sampling algorithms such as Q-learning, as it is not necessary to have a perfect knowledge about the environment - particularly the transition probabilities between states are often unavailable in real-world environments. In Chapter 4 we first describe the standard Q-learning algorithm and touch on convergence conditions.

The new and improved CVaR Value Iteration procedure opens a door for a sampling version of the algorithm which is our second original contribution.

2. **CVaR Q-learning:** Using methods of recursive VaR-CVaR estimation, we formulate a Temporal Difference update equation, based on the improved CVaR Value Iteration, that finds the optimal value function in expectation and formulate a new algorithm called CVaR Q-learning.

   We then experimentally verify the correctness of our approach by showing that the algorithm learns risk-sensitive policies on different confidence levels.

While in standard reinforcement learning it is straightforward to extract the optimal policy once an optimal value function has been learned, this is not the case in CVaR Q-learning, due to the time-inconsistency of the CVaR criterion.

3. **CVaR policy improvement:** We propose a policy improvement algorithm for distributional reinforcement learning and prove its correctness. This procedure is then used as a consistent heuristic for extracting the optimal policy from CVaR Q-learning, and we empirically show its validity when used in conjunction with linear interpolation.

The holy grail of reinforcement learning is the ability to successfully learn in vast state spaces. The methods proposed in this thesis should also ultimately be usable on large state spaces where exact Q-learning becomes intractable, and this is something we explore in Chapter 5. We start with a brief introduction to deep learning followed by deep Q-learning. We also touch on quantile regression Q-learning, a recent distributional reinforcement learning algorithm, that serves as an introduction to the concepts explored later.

4. **Deep CVaR Q-learning:** We extend CVaR Q-learning to its approximate variant by formulating the Temporal Difference update rule as arguments to minimizing the $\mathcal{L}_{\text{VaR}}$ and $\mathcal{L}_{\text{CVaR}}$ loss functions. We then combine the loss functions with the well-known DQN [40] algorithm and show that the new Deep CVaR Q-learning algorithm is capable of learning risk-sensitive policies from raw pixels, hereby demonstrating the scalability and practicality of proposed approaches.

2

PRELIMINARIES

The goal of this chapter is to provide a formal background on the covered material, together with a unified notation (which differs quite a lot from publication to publication). After we establish some basics of reinforcement learning in Section 2.1, we follow up with the recently explored and useful distributional reinforcement learning in Section 2.2. Next we tackle the basics of risk together with the crucial CVaR measure in Section 2.3.

The interested reader is welcome to explore the books and publications referenced throughout this chapter and in Section 2.5. An informed reader may choose to skip to Section 2.4 where we formalize the problems tackled in this thesis.

2.1 REINFORCEMENT LEARNING

The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. Reinforcement learning [53] is a sub-field of machine learning that deals with time-dependent decision making in an unknown environment. The learner (often called agent) is not told which actions to take, but instead must discover which actions yield the most reward by trying them out. In the most interesting and challenging cases, actions may affect not only the immediate reward but also subsequent situations and rewards. These two characteristics, trial-and-error search and delayed reward are the most important distinguishing features of reinforcement learning.

The general interaction between the agent and an environment can be seen in Figure 2.1. In each time-step t, the agent receives an observation x_t and a reward r_t and picks an action a_t and the process repeats. Below we formalize all the necessary notions of states, actions and rewards as a Markov Decision Process.

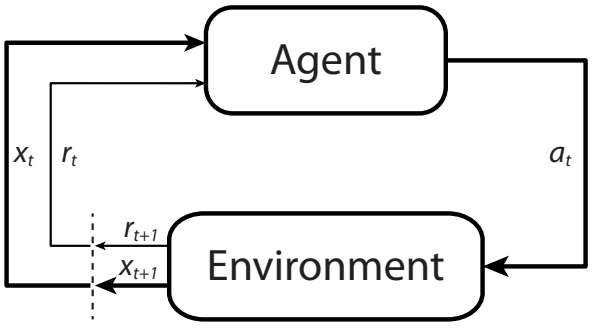


Figure 2.1: The Reinforcement learning cycle

2.1.1 *Markov Decision Processes*

Markov Decision Process (MDP, Bellman [12]) is a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards. They are a mathematically idealized form of the reinforcement learning problem for which precise theoretical statements can be made.

The word Markov points to the fact that we assume that the state transitions of an MDP satisfy the memoryless property. This means that the conditional probability distribution of future states of the process depends only upon the present state and not the whole history of events that preceded it.

Definition. *MDP is a 5-tuple* $\mathcal{M} = (\mathcal{X}, \mathcal{A}, r, p, \gamma)$*, where*
$\mathcal{X}$ *is the finite state space*
$\mathcal{A}$ *is the finite action space*
$r(x, a)$ *is a bounded deterministic*[1] *reward generated by being in state* x *and selecting action* a
$p(\cdot|x, a)$ *is the transition probability distribution*
$\gamma \in [0, 1)$ *is a discount factor*

We will denote x or x_t as the states visited in time t and x' or x_{t+1} states visited in time $t + 1$.

Definition. *A stationary (or Markovian) policy is a mapping from states to actions* $\pi : \mathcal{X} \to \mathcal{A}$.

2.1.2 *Return*

The ultimate goal of any reinforcement learning agent is to maximize some notion of reward, which is captured by the return. The two most commonly considered types of returns are the sum of rewards, and the mathematically convenient expected discounted reward. In this thesis we focus on the latter.

We define the return $Z^\pi(x)$ as a random variable representing the discounted reward along a trajectory generated by the MDP by following policy π, starting at state x:

$$\begin{aligned} &Z^\pi(x) = \sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \\ &x_t \sim p\big(\cdot|x_{t-1}, a_{t-1}\big), a_t \sim \pi, x_0 = x \end{aligned} \tag{2.1}$$

As a useful notation, we denote $Z^\pi(x, a)$ as the random variable representing the discounted reward along a trajectory generated by first selecting action a and then following policy π.

$$\begin{aligned} &Z^\pi(x, a) = \sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \\ &x_t \sim p\big(\cdot|x_{t-1}, a_{t-1}\big), a_t \sim \pi, x_0 = x, a_0 = a \end{aligned} \tag{2.2}$$

1 All the presented results are extendable to the case with random reward and with reward being a function of the transition $R(x, a, a')$. In fact, we use $R(x, a, a')$ in all our experiments. We avoid these extension for the sake of cleaner notation.

We will sometimes omit the superscript π when the policy is clear from context or is unimportant.

2.1.3 *Bellman equations*

The *value function* $V^\pi : \mathcal{X} \to \mathbb{R}$ of policy π describes the expected return received from state $x \in \mathcal{X}$ and acting according to π:

$$V^\pi(x) = \mathbb{E}\left[Z^\pi(x)\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(x_t, a_t)\right] \\ x_0 = x, a_t \sim \pi \tag{2.3}$$

The *action-value function* $Q^\pi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ of policy π describes the expected return of taking action $a \in \mathcal{A}$ in state $x \in \mathcal{X}$, then acting according to π:

$$Q^\pi(x, a) = \mathbb{E}\left[Z^\pi(x, a)\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(x_t, a_t)\right] \\ x_0 = x, a_0 = a, a_t \sim \pi \tag{2.4}$$

Fundamental to reinforcement learning is the use of Bellman's equation [12] to describe the value and action-value functions by a recursive relationship:

$$V^\pi(x) = r\left(x, \pi(x)\right) + \gamma \mathop{\mathbb{E}}_{p,\pi}\left[V^\pi(x')\right] \tag{2.5}$$

$$Q^\pi(x, a) = r(x, a) + \gamma \mathop{\mathbb{E}}_{p,\pi}\left[V^\pi(x')\right] \tag{2.6}$$

As stated before, we are typically interested in maximizing the expected return. The most common approach for doing so involves the optimality equation

$$Q^*(x, a) = r(x, a) + \gamma\, \mathbb{E}_p\left[\max_{a' \in \mathcal{A}} Q^*(x', a')\right] \tag{2.7}$$

This equation has a unique fixed point Q^* - the optimal value function, corresponding to the set of optimal policies Π^* (π^* is optimal if $\mathbb{E}_{a \sim \pi^*} Q^*(x, a) = \max_a Q^*(x, a)$). We view value functions as vectors in $\mathbb{R}^{\mathcal{X} \times \mathcal{A}}$, and the optimal value function as one such vector. In this context, the *Bellman operator* $\mathcal{T}^\pi$ and the *optimality operator* $\mathcal{T}$ are

$$\mathcal{T}^\pi Q(x, a) := r(x, a) + \gamma \mathop{\mathbb{E}}_{P,\pi}\left[Q(x', a')\right] \tag{2.8}$$

$$\mathcal{T} Q(x, a) := r(x, a) + \gamma \mathop{\mathbb{E}}_{P}\left[\max_{a' \in \mathcal{A}} Q(x', a')\right] \tag{2.9}$$

These operators are useful as they describe the expected behaviour of popular learning algorithms such as SARSA and Q-Learning [53]. In particular they are both contraction mappings (see below), and their repeated application to some initial Q_0 converges exponentially to Q^π or Q^*, respectively [14].

2.1.4 *Contraction*

An important concept used in convergence analysis of reinforcement learning algorithms is that of a contraction.

Definition. *A **fixed point** of a mapping $T : S \to S$ of a set S into itself is $s \in S$ which is mapped onto itself, that is $Ts = s$.*

*Let $S = (S, d)$ be a metric space (d is a metric on S). A mapping T is called a **contraction** on S if there exists a positive real number $\gamma < 1$ such that for all $s, t \in S$ we have $d(Ts, Tt) \leq \gamma d(s, t)$.*

Theorem 1 (Banach Fixed Point Theorem, Contraction Theorem)**.** *Consider a metric space $S = (S, d)$, where $S \neq \emptyset$. Suppose that S is complete (every Cauchy sequence in S converges) and let T be a contraction on S. Then T has precisely one fixed point.*

The takeaway for reinforcement learning is, that if $\mathcal{T}$ is a contraction, by recursively applying $V_{t+1} = \mathcal{T}V_t$ (here the operator maps value function $V_t \in \mathbb{R}$ to a new $V_{t+1} \in \mathbb{R}$), we converge to the single fixed point of this operator. This is used in convergence proofs for various RL operators, usually in combination with the infinity norm.

See e.g. Kreyszig [34] for a complete treatment of all mentioned concepts.

2.2 DISTRIBUTIONAL REINFORCEMENT LEARNING

In contrast to standard reinforcement learning, where we model the expected return, in distributional reinforcement learning we aim to model the full distribution of the return. This is advantageous in cases where we want to e.g. model parametric uncertainty or design risk-sensitive algorithms [41, 42]. Bellemare, Dabney, and Munos [10] also argue, that the distributional approach is beneficial even in the case when we optimize the expected value, as the distribution gives us more information which helps the now commonly used approximate algorithms (such as DQN [40]).

At the core of the distributional approach lies the recursive equation of the return distribution:

$$\begin{aligned} &Z(x, a) \overset{D}{=} r(x, a) + \gamma Z(x', a') \\ &x' \sim p(\cdot|x, a), a \sim \pi, x_0 = x, a_0 = a \end{aligned} \tag{2.10}$$

where $\overset{D}{=}$ denotes that random variables on both sides of the equation share the same probability distribution.

In the *policy evaluation* setting [53] we are interested in the value function V^π associated with a given policy π. The analogue here is the value distribution Z^π. Note that Z^π describes the intrinsic randomness of the agent's interactions with its environment, rather than some measure of uncertainty about the environment itself.

We define the transition operator $P^\pi : \mathcal{Z} \to \mathcal{Z}$ as

$$\begin{aligned} &P^\pi Z(x, a) \overset{D}{=} Z(x', a') \\ &x' \sim p(\cdot|x, a), a' \sim \pi \end{aligned} \tag{2.11}$$

and the distributional Bellman operator $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ as

$$\mathcal{T}^\pi Z(x, a) \overset{D}{=} r(x, a) + \gamma P^\pi Z(x, a). \tag{2.12}$$

Note that this is a distributional equation and the distributional bellman operator is therefore fundamentally different from the standard bellman operator.

Bellemare, Dabney, and Munos [10] have shown, that the distributional bellman operator $\mathcal{T}^\pi$ is not a contraction in the commonly used KL divergence [36], but is a contraction in the infinity Wasserstein metric which we describe bellow, as it will become useful as a tool for evaluating algorithms in the rest of the thesis. Another important fact is, that the bellman optimality operator $\mathcal{T} : \mathcal{Z} \to \mathcal{Z}$

$$\mathcal{T}Z = \mathcal{T}^\pi Z \quad \text{for some } \pi \text{ in the set of greedy policies} \tag{2.13}$$

is not a contraction in any metric. The distribution does not converge to a fixed point, but rather to a sequence of optimal (in terms of expected value) policies. We encourage the interested reader to explore these topics in the excellent papers by Bellemare, Dabney, and Munos [10] and Dabney et al. [23].

2.2.1 *The Wasserstein Metric*

One of the tools for analysis of distributional approaches to reinforcement learning is the Wasserstein metric d_p between cumulative distribution functions (see e.g. Bickel and Freedman [15]). The metric has recently gained in popularity and was used e.g. in unsupervised learning [3, 11]. Unlike the Kullback-Leibler divergence, which strictly measures change in probability, the Wasserstein metric reflects the underlying geometry between outcomes.

For F, G two c.d.fs over the reals, it is defined as

$$d_p(F, G) := \inf_{U,V} \|U - V\|_p,$$

where the infimum is taken over all pairs of random variables (U, V) with respective cumulative distributions F and G. The infimum is attained by the inverse c.d.f. transform of a random variable $\mathcal{U}$ uniformly distributed on $[0, 1]$:

$$d_p(F, G) = \|F^{-1}(\mathcal{U}) - G^{-1}(\mathcal{U})\|_p.$$

For $p < \infty$ this is more explicitly written as

$$d_p(F, G) = \left(\int_0^1 \left| F^{-1}(u) - G^{-1}(u) \right|^p du \right)^{1/p}. \tag{2.14}$$

meaning it is an integral over the difference in the quantile functions of the random variables. This will become important later, as the quantile function has a direct connection to the CVaR objective (2.21) explored in this thesis.

2.3 RISK-SENSITIVITY

The standard reinforcement learning agent that maximizes the expected reward which we discussed in the previous chapter does not take risk into account. Indeed in a mathematical sense, the shape of the reward distribution is unimportant as wildly different distributions may have the same expectation. This unfortunately is not the case in the real world, where there exist catastrophic losses - an investment company may have a good strategy that yields profit in expectation, but if the strategy is too risky and the company's capital drops under zero, the investment strategy is

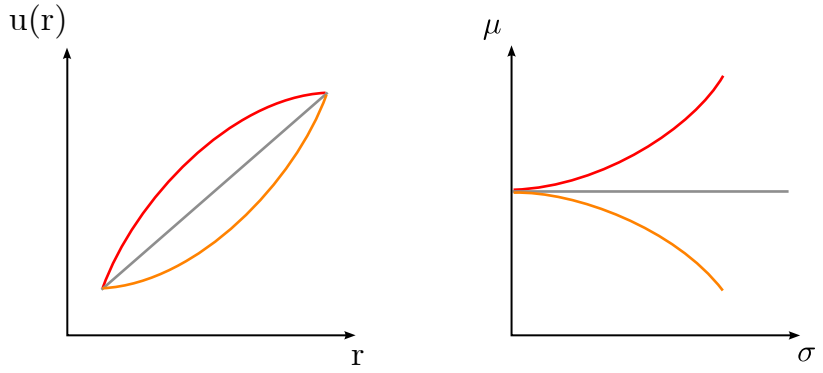


Figure 2.2: Standard in economic literature, these figures depict the differences between risk-averse (red), risk-neutral (grey) and risk-seeking (orange) behaviors.
Left: A subjective utility $u(r)$, based on the reward, is concave for risk-averse behaviors. Right: Risk-averse utility contour lines in standard deviation-expected value space are upward sloped.

useless. This leads us to defining risk, which describes the potential of gaining or losing reward and is therefore more expressive than a simple expectation.

The finance literature differentiates between three risk-related types of behavior, namely *risk-neutral*, *risk-averse* and *risk-seeking*. We offer the following example to illustrate the differences between mentioned behaviors: Imagine you are facing two options, either (a) you get \$1 or (b) you get \$5 with 90% probability, but lose \$35 with 10% probability. A risk-neutral agent wouldn't differentiate between the two choices, as the expected value of reward is the same. A risk-averse agent would prefer option (a), as there is a risk of high losses in option (b). Risk-seeking agent would pick (b). The difference between the risk-sensitive behaviors can be visualized as in Figure 2.2.

The desired behavior for most critical applications is risk-averse and indeed it is the behavior of choice for financial institutions [21, 60]. It has also been suggested that humans prefer risk-averse behaviors when making decisions [50].

As we stated in the introduction, we are interested in reinforcement learning that maximizes a certain risk-averse objective. Below we formally describe the metrics used to measure risk which we then use to formulate the exact problem tackled in this thesis.

2.3.1 *Value-at-Risk*

Value-at-risk (VaR, see e.g. Wipplinger [60]) is one of the most popular tools used to estimate exposure to risk used risk management and financial control.

Let Z be a random variable representing reward, with cumulative distribution function (c.d.f.) $F(z) = \mathbb{P}(Z \leq z)$. The Value-at-Risk at confidence level $\alpha \in (0, 1)$ is the α-quantile of Z, i.e.

$$\mathrm{VaR}_\alpha(Z) = F^{-1}(\alpha) = \inf\{z | \alpha \leq F(z)\} \tag{2.15}$$

We will use the notation $\mathrm{VaR}_\alpha(Z)$, $F^{-1}(\alpha)$ interchangeably, often explicitly denoting the random variable of inverse c.d.f. as $F_Z^{-1}(\alpha)$.

Note on notation: In the risk-related literature, it is common to work with losses instead of rewards. The Value-at-Risk is then defined as the $1-\alpha$ quantile. The notation we use reflects the use of reward in reinforcement learning and this sometimes leads to the need of reformulating some definitions or theorems. While these reformulations may differ in notation, they characterize the same underlying ideas.

2.3.2 *Conditional Value-at-Risk*

Conditional Value-at-Risk (CVaR, see Rockafellar and Uryasev [47, 48]), sometimes called Expected Shortfall (ES), Average Value-at-Risk (AVaR) or Tail Value-at-Risk (TVaR), is a risk measure that aims to fix inadequacies of measuring risk introduced by Value-at-Risk. Firstly, it has the desirable mathematical properties of monotonicity, translation invariance, positive homogeneity and subadditivity (see Artzner et al. [4]), which makes CVaR computation much easier compared to VaR. its properties were also recently identified as suitable for measuring risk in robotics [38]. Another strong point of CVaR is that unlike VaR, it is able to distinguish between large and catastrophic losses. For these reasons, CVaR is starting to replace VaR as a standard measure for risk in financial applications [21] and beyond.

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

$$\text{CVaR}_\alpha(Z) = \frac{1}{\alpha}\int_0^\alpha F_Z^{-1}(\beta)\mathrm{d}\beta = \frac{1}{\alpha}\int_0^\alpha \text{VaR}_\beta(Z)\mathrm{d}\beta \tag{2.16}$$

Rockafellar and Uryasev [47] also showed that CVaR is equivalent to the solution of the following optimization problem

$$\text{CVaR}_\alpha(Z) = \max_s \left\{ \frac{1}{\alpha}\mathbb{E}\left[(Z-s)^-\right] + s \right\} \tag{2.17}$$

where $(x)^- = \min(x, 0)$ represents the negative part of x and in the optimal point it holds that $s^* = VaR_\alpha(Z)$

$$\text{CVaR}_\alpha(Z) = \frac{1}{\alpha}\mathbb{E}\left[(Z - VaR_\alpha(Z))^-\right] + VaR_\alpha(Z) \tag{2.18}$$

The last definition we will need is the dual formulation of (2.17)

$$\text{CVaR}_\alpha(Z) = \min_{\xi \in \mathcal{U}_{\text{CVaR}}(\alpha, p(\cdot))} \mathbb{E}_\xi[Z] \tag{2.19}$$

$$\mathcal{U}_{\text{CVaR}}(\alpha, p(\cdot)) = \left\{ \xi : \xi(z) \in \left[0, \frac{1}{\alpha}\right], \int \xi(z)p(z)\mathrm{d}z = 1 \right\} \tag{2.20}$$

We provide basic intuition behind the dual variables as these will become important later: Since we are minimizing the variables ξ over the set $\mathcal{U}_{\text{CVaR}}(\alpha, p(\cdot))$, the optimal values are $\xi(z) = \min\left(\frac{1}{\alpha}, \frac{1}{p(z)}\right)$ for the lowest possible values z, as these values influence the resulting $\text{CVaR}_\alpha(Z)$. Values above $\text{VaR}_\alpha(Z)$ are not taken into account so their ξ is 0. If there exists a discrete atom at $\text{VaR}_\alpha(Z)$, the variables are linearly interpolated to fit the constraints.

For a concise treatment of duality, see e.g. Boyd and Vandenberghe [18].

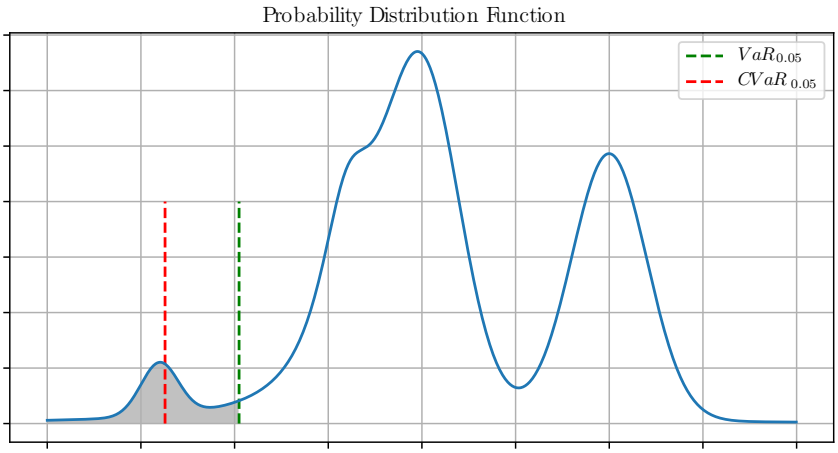


Figure 2.3: Value-at-Risk and Conditional Value-at-Risk of a general probability distribution with the integral $\alpha = 0.05$ marked in grey. The main flaw of the VaR metric is clearly visible here, as we could shift the leftmost 'mode' of the distribution into minus infinity and the VaR would remain unchanged, while CVaR would change with the shift.

2.4 PROBLEM FORMULATION

The problem tackled in this thesis considers reinforcement learning with optimization of the CVaR objective. Unlike the expected value criterion, it is insufficient to consider only stationary policies, and we must work with general history-dependent policies. We define them formally below.

Definition (History-Dependent Policies)**.** *Let the space of admissible histories up to time t be $H_t = H_{t-1} \times \mathcal{A} \times \mathcal{X}$ for $t \geq 1$, and $H_0 = \mathcal{X}$. A generic element $h_t \in H_t$ is of the form $h_t = (x_0, a_0, ..., x_{t-1}, a_{t-1})$. Let $\Pi_{H,t}$ be the set of all history-dependent policies with the property that at each time t the randomized control action is a function of h_t. In other words, $\Pi_{H,t} = \{\pi_0 : H_0 \to \mathbb{P}(\mathcal{A}), ..., \pi_t : H_t \to \mathbb{P}(\mathcal{A})\}$. We also let $\Pi_H = \lim_{t\to\infty} \Pi_{H,t}$ be the set of all history-dependent policies.*

The risk-averse objective we wish to address for a given confidence level α is

$$\max_{\pi \in \Pi_H} \text{CVaR}_\alpha(Z^\pi(x_0)) \tag{2.21}$$

where $Z^\pi(x_0)$ coincides with definition (2.1).

In words, our goal is to find a general policy $\pi^* \in \Pi_H$, that maximizes Conditional Value-at-Risk of the return, starting in state x_0. We emphasize the importance of the starting state since, unlike the expected value, the CVaR objective is not time-consistent.

2.4.1 *Time-consistency*

There exist several definitions of time-consistency [16, 43]. Informally, if the criterion is time-consistent, we can limit ourselves to the space of stationary policies, as the

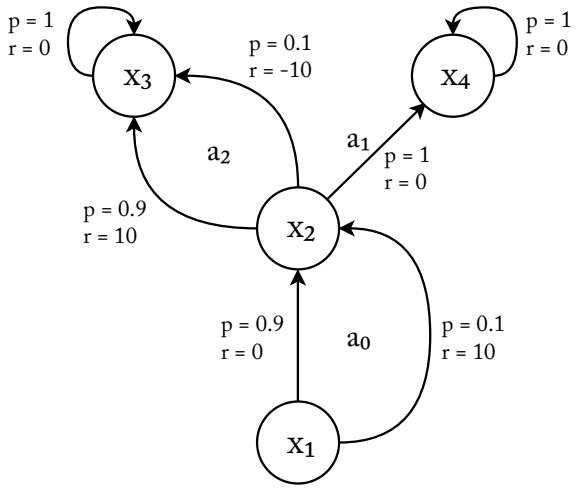


Figure 2.4: MDP showing time-inconsistency of the CVaR objective.

optimal policy is part of this space. On the other hand, non-stationary policies may be required to solve a time-inconsistent problem.

We provide the following example to show that the CVaR criterion is indeed not time-consistent. In Figure 2.4 we can see an MDP with starting state x_1 with a single action a_0, followed by state x_2 where the agent can choose between actions a_1, a_2; states x_3, x_4 are terminal. We now compare three policies $\pi_1(x_1) = a_1, \pi_2(x_1) = a_2$ and a non-stationary policy π_3 that chooses a_2, unless the agent was lucky and received the reward 10 when transitioning from state x_1. Let us examine the CVaR objective with $\alpha = 0.19$ and $\gamma = 1$ (or close to 1):

$$\begin{aligned}
\text{CVaR}_{0.19}(Z^{\pi_1}(x_1)) &= \frac{0.19 \cdot 0}{0.19} = 0 \\
\text{CVaR}_{0.19}(Z^{\pi_2}(x_1)) &= \frac{0.09 \cdot (-10) + 0.01 \cdot 0 + 0.09 \cdot 10}{0.19} = 0 \\
\text{CVaR}_{0.19}(Z^{\pi_3}(x_1)) &= \frac{0.09 \cdot (-10) + 0.1 \cdot 10}{0.19} \doteq 0.526
\end{aligned}$$

By examining the results, we can see that the non-stationary policy π_3 is better than any stationary one, confirming CVaR as a time-inconsistent objective, explaining the need for a history-dependent policy in our problem definition 2.21.

As we will see later, the time-inconsistency can be sidestepped by extending the state space by a continuous parameter $y \in [0, 1]$ which represents the different confidence levels we may choose to optimize. Notably, the optimal policy must in this case look at different confidence levels in each state - otherwise we would consider only stationary policies.

2.4.2 *Robustness*

An important motivational point for the CVaR objective (2.21) is its relationship with robustness. Chow et al. [20] have shown that optimizing the objective is equivalent to being robust to model perturbations. Thus, by minimizing CVaR, the decision

maker also guarantees robustness to modeling errors. For completeness, we repeat the formulation of the equivalence relation below.

Let $(x_0, a_0, ..., x_T)$ be a trajectory in a finite-horizon MDP problem. The total probability of the trajectory is $p(x_0, a_0, ..., x_T) = p(x_0)p(x_1|x_0, a_0) \cdots p(x_T|x_{T-1}, a_{T-1})$. For each step $1 \leq t \leq T$ consider a perturbed transition matrix $\hat{P} = P \circ \delta_t$ where $\delta_t \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}\times\mathcal{X}}$ and $\circ$ is the element-wise product under the condition that $\hat{P}$ is a stochastic matrix. Let Δ_t be the set of perturbation matrices that satisfy this condition and $\Delta = \Delta_1 \times \cdots \times \Delta_T$ be the set of all possible perturbations to the trajectory distribution.

We now impose a budget constraint on the perturbations as follows. For some budget $\eta \geq 1$, we consider the constraint

$$\delta_1(x_1|x_0)\delta_2(x_2|x_1) \cdots \delta_T(x_T|x_{T-1}) \leq \eta \tag{2.22}$$

Essentially, the product in (2.22) states that *the worst cannot happen at each time*. Instead, the perturbation budget has to be split (multiplicatively) along the trajectory. We note that (2.22) is in fact a constraint on the perturbation matrices, and we denote by $\Delta_\eta \subset \Delta$ the set of perturbations that satisfy this constraint with budget η. Then the following holds (Proposition 1 of [20])

$$\mathrm{CVaR}_{\frac{1}{\eta}}\left(R_{0,T}(x_1, \ldots, x_T)\right) = \inf_{(\delta_1,\ldots,\delta_T)\in\Delta_\eta} \mathbb{E}_{\hat{P}}[R_{0,T}(x_1, \ldots, x_T)], \tag{2.23}$$

where $R_{0,T}(x_1, \ldots, x_T)$ denotes the random variable representing the reward along the particular trajectory and $\mathbb{E}_{\hat{P}}[\cdot]$ denotes expectation with respect to a Markov chain with transitions $\hat{P}_t$.

2.5 LITERATURE SURVEY

Risk-sensitive MDPs have been studied thoroughly in the past, with different risk-related objectives. Due to its good computational properties, earlier efforts focused on exponential utility [28], the max-min criterion [22] or e.g. maximizing the mean with constrained variance [52]. A comprehensive overview of the different objectives can be found in Garcıa and Fernández [25], together with a unified look on the different methods used in safe reinforcement learning. Among CVaR-related objectives, some publications focus on optimizing the expected value with a CVaR constraint [17, 45].

Recently, for the reasons explained above, several authors have investigated the exact objective (2.21). A considerable effort has gone towards policy-gradient [54] and Actor-Critic [33] algorithms with the CVaR objective. Chow and Ghavamzadeh [19] and Tamar, Glassner, and Mannor [55] present useful ways of computing the CVaR gradients with parametric models and have shown the practicality and scalability of these approaches on interesting domains such as the well-known game of Tetris. An important setback of these methods is their limitation of the hypothesis space to the class of stationary policies, meaning they can only reach a *local* minimum of our objective. Similar policy gradient methods have also been investigated in the context of general coherent measures, a class of risk measures encapsulating many used measures including CVaR. Tamar et al. present a policy gradient algorithm and a gradient-based Actor-Critic algorithm [56]. Again, these algorithms only converge in local extremes.

Some authors have also tried to sidestep the time-consistency issue of CVaR by either focusing on a time-consistent subclass of coherent measures, limiting the hypothesis space to time-consistent policies, or reformulating the CVaR objective in a time-consistent way [39].

Morimura et al. [41, 42] were among the first to utilize distributional reinforcement learning with both parametric and nonparametric models and used it to optimize CVaR. They only used the naive approach discussed in Section 2.4.1 in their experiments.

Bäuerle and Ott [8] use a state space extensions and show that this new extended state space contains globally optimal policies. Unfortunately, because the state-space is continuous, the design of a solution algorithm is challenging.

The approach of Chow et al. [20] also uses a continuous augmented state-space but unlike Bäuerle and Ott [8], this continuous state is shown to have bounded error when a particular linear discretization is used. The only flaw of this approach is the requirement of running a linear program in each step of their algorithm and we address this issue in the next chapter.

3

VALUE ITERATION WITH CVAR

Value iteration is a standard algorithm for maximizing the expected discounted reward used in reinforcement learning. In this chapter we extend the results of Chow et al. [20], who have recently proposed an approximate value iteration algorithm for CVaR MDPs.

The original algorithm requires a computation of a linear program in each step of the value iteration procedure. Utilizing a connection between the used αCVaR_α function and the quantile function, we sidestep the need for this computation and propose a linear-time version of the algorithm, making CVaR value iteration feasible for much larger MDPs.

After reminding the reader of the standard value iteration, we present the CVaR Value Iteration algorithm in Section 3.2. Our faster approach is presented in Section 3.3, followed by section Section 3.4, where we test the new method on selected environments.

3.1 VALUE ITERATION

Value iteration [53] is a well-known algorithm for computing the optimal action-value function and hereby finding an optimal policy. Let us remind ourselves of the Bellman optimality operator $\mathcal{T}$ (2.9):

$$\mathcal{T}Q(x,a) := r(x,a) + \gamma \sum_{x'} p(x'|x,a) \max_{a' \in \mathcal{A}} Q(x',a')$$

or rewritten for the value function V

$$\mathcal{T}V(x) = \max_a \left\{ r(x,a) + \gamma \sum_{x'} p(x'|x,a) V(x') \right\} \tag{3.1}$$

As stated before, $\mathcal{T}$ is a contraction (Section 2.1.4). This means that by repeatedly applying the operator we eventually converge to the optimal point, since we converge and the definition holds in this point. This leads to the formulation of the *Value Iteration* algorithm. The only difference between theory and practice is the introduction of a small parameter ϵ that allows us to check the converge and end the algorithm when we reach a certain precision, as the contraction converges only in the limit.

See Algorithm 1.

Algorithm 1 Value Iteration

Initialize V arbitrarily (e.g. $V(x) = 0$ for all $x \in \mathcal{X}$)
repeat
  $v = V(x)$
  $\Delta = 0$
  for each $x \in \mathcal{X}$ **do**
    $V(x) = \max_a \{r(x,a) + \gamma \sum_{x'} p(x'|x,a)V(x')\}$
    $\Delta = \max\{\Delta, |v - V(x)|\}$
  end for
until $\Delta < \epsilon$
Output a deterministic policy $\pi \approx \pi^*$:
  $\pi(x) = \arg\max_a \{r(x,a) + \gamma \sum_{x'} p(x'|x,a)V(x')\}$

3.2 CVAR VALUE ITERATION

Chow et al. [20] present a dynamic programming formulation for the CVaR MDP problem (2.21). As CVaR is a time-inconsistent measure, their method requires an extension of the state space. A Value Iteration type algorithm is then applied on this extended space and Chow et al. [20] proved its convergence.

We repeat their key ideas and results bellow, as they form a basis for our contributions presented in later sections. The results are presented with our notation introduced in Chapter 2, which differs slightly from the paper, but the core ideas remain the same.

3.2.1 *Bellman Equation for CVaR*

The results of Chow et al. [20] heavily rely on the CVaR decomposition theorem (Lemma 22, [43]):

$$\begin{aligned} \mathrm{CVaR}_\alpha\left(Z^\pi(x)\right) &= \min_{\xi \in \mathcal{U}_{\mathrm{CVaR}}(\alpha, p(\cdot|x,a))} \sum_{x'} p(x'|x,\pi(x))\xi(x')\mathrm{CVaR}_{\xi(x')\alpha}\left(Z^\pi(x')\right) \\ \mathcal{U}_{\mathrm{CVaR}}(\alpha, p(\cdot|x,a)) &= \left\{\xi : \xi(z) \in \left[0, \frac{1}{\alpha}\right], \int \xi(z)p(z)\mathrm{d}z = 1\right\} \end{aligned} \tag{3.2}$$

where the risk envelope $\mathcal{U}_{\mathrm{CVaR}}(\alpha, p(\cdot|x,a))$ coincides with the dual definition of CVaR (2.20).

The theorem states that we can compute the $\mathrm{CVaR}_\alpha\left(Z^\pi(x,a)\right)$ as the minimal weighted combination of $\mathrm{CVaR}_\alpha\left(Z^\pi(x')\right)$ under a probability distribution perturbed by $\xi(x')$. Notice that the variable ξ appears in both the sum *and* modifies the confidence level for each state.

Also note that the decomposition requires only the representation of CVaR at different confidence levels and not the whole distribution at each level, which we might be tempted to think because of the time-inconsistency issue.

Chow et al. [20] extend the decomposition theorem by defining the *CVaR value function*[1] $C(x, y)$ with an augmented state-space $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{Y} = (0, 1]$ is an additional continuous state that represents the different confidence levels.

$$C(x, y) = \max_{\pi \in \Pi_H} \text{CVaR}_y \left(Z^\pi(x) \right) \tag{3.3}$$

Similar to standard dynamic programming, it is convenient to work with with operators defined on the space of value functions. This leads to the following definition of the CVaR Bellman operator $\mathbf{T} : \mathcal{X} \times \mathcal{Y} \to \mathcal{X} \times \mathcal{Y}$:

$$\mathbf{T}C(x, y) = \max_a \left[r(x, a) + \gamma \min_{\xi \in \mathcal{U}_{\text{CVaR}}(\alpha, p(\cdot|x,a))} \sum_{x'} p(x'|x, a)\xi(x')C\left(x', y\xi(x')\right) \right] \tag{3.4}$$

or in our simplified notation, this describes the following relationship:

$$\mathbf{T}\text{CVaR}_y(Z(x)) = \max_a \left[r(x, a) + \gamma \text{CVaR}_y(P^{\pi^*} Z(x, a)) \right] \tag{3.5}$$

where P^π denotes the transition operator (2.11).

Chow et al. [20] further showed (Lemma 3) that the operator $\mathbf{T}$ is a contraction and also preserves the convexity of $y\text{CVaR}_y$. The optimization problem (3.2) is a convex one and therefore has a single solution. Additionally, the fixed point of this contraction is the optimal $C^*(x, y) = \max_{\pi \in \Pi} \text{CVaR}_y(Z^\pi(x, y))$ ([20], Theorem 4).

Naive value iteration with operator $\mathbf{T}$ is unfortunately unusable in practice, as the state space is continuous in y. The solution proposed in [20] is then to represent the convex $y\text{CVaR}_y$ as a piecewise linear function.

3.2.2 *Value Iteration with Linear Interpolation*

Given a set of $N(x)$ interpolation points $\mathbf{Y}(x) = \left\{ y_1, \ldots, y_{N(x)} \right\}$, we can approximate the $yC(x, y)$ function by interpolation on these points, i.e.

$$\mathcal{I}_x[C](y) = y_i C(x, y_i) + \frac{y_{i+1} C(x, y_{i+1}) - y_i C(x, y_i)}{y_{i+1} - y_i}(y - y_i),$$

where $y_i = \max \{ y' \in \mathbf{Y}(x) : y' \leq y \}$. The interpolated Bellman operator $\mathbf{T}_\mathcal{I}$ is then also a contraction and has a bounded error ([20], Theorem 7).

$$\mathbf{T}_\mathcal{I} C(x, y) = \max_a \left[r(x, a) + \gamma \min_{\xi \in \mathcal{U}_{\text{CVaR}}(\alpha, p(\cdot|x,a))} \sum_{x'} p(x'|x, a) \frac{\mathcal{I}_{x'}[C](y\xi(x'))}{y} \right] \tag{3.6}$$

The full value iteration procedure is presented in Algorithm 2.

This algorithm can be used to find an approximate global optimum in any MDP. There is however the issue of computational complexity. As the algorithm stands, the straightforward approach is to solve each iteration of (3.6) as a linear program, since the problem is convex and piecewise linear, but this is not practical, as the LP computation can be demanding and is therefore not suitable for large state-spaces.

For completeness, we formulate the full linear program in the appendix (A.1).

1 We use C instead of V in our notation.

Algorithm 2 CVaR Value Iteration with Linear Interpolation (Algorithm 1 in [20])

1: **Given:**

- $N(x)$ interpolation points $\mathbf{Y}(x) = \left\{y_1, \ldots, y_{N(x)}\right\} \in [0,1]^{N(x)}$ for every $x \in \mathcal{X}$ with $y_i < y_{i+1}$, $y_0 = 0$ and $y_{N(x)} = 1$.
- Initial value function $C_0(x, y)$ that satisfies:
  1. $yC_0(x, y)$ is convex in y for all x
  2. $yC_0(x, y)$ is continuous in y for all x

2: Repeat until convergence:

- For each $x \in \mathcal{X}$ and each $y_i \in \mathbf{Y}(x)$, update the value function estimate as follows:

$$C_{k+1}(x, y_i) = \mathbf{T}_{\mathcal{I}}[C_k](x, y_i),$$

3: Set the converged value iteration estimate as $\widehat{C}^*(x, y_i)$, for any $x \in \mathcal{X}$, and $y_i \in \mathbf{Y}(x)$.

3.2.3 *Optimal policy*

An important product of any value iteration algorithm is the optimal policy. Since we know that the CVaR objective isn't time-consistent, we must change the optimized confidence level in each state with each sampled transition. The value-function C^* can be used to extract the optimal policy π^* of the original problem (2.21), using the following theorem.

Theorem 2 (Optimal Policies, Theorem 5 in [20])**.** *Let $\pi_H^* = \{\mu_0, \mu_1, \ldots\} \in \Pi_H$ be a history-dependent policy recursively defined as:*

$$\mu_t(h_t) = u^*(x_t, y_t), \ \forall k \geq 0, \tag{3.7}$$

with initial conditions x_0 and $y_0 = \alpha$, and state transitions

$$x_t \sim P(\cdot \mid x_{k-1}, u^*(x_{k-1}, y_{k-1})), \quad y_t = y_{k-1}\xi^*_{x_{k-1}, y_{k-1}, u^*}(x_t), \forall k \geq 1, \tag{3.8}$$

where the stationary Markovian policy $u^*(x, y)$ and risk factor $\xi^*_{x,y,u^*}(\cdot)$ are solution to the max-min optimization problem in the CVaR Bellman operator $\mathbf{T}[C^*](x, y)$. Then, π_H^* is an optimal policy for problem (2.21) with initial state x_0 and CVaR confidence level α.

And the theorem holds for both the original operator $\mathbf{T}$ and the linearly interpolated $\mathbf{T}_{\mathcal{I}}$.

Chow et al. [20] further showed that the error is bounded and linearly dependent on parameter θ, where θ describes a logarithmic rule with which we select atoms $y_{i+1} = \theta y_i$.

3.3 EFFICIENT COMPUTATION USING QUANTILE REPRESENTATION

We present our original contributions in this section, first describing a connection between the $y\text{CVaR}_y$ function and the quantile function of the underlying distribution. We then use this connection to formulate a faster computation of the value iteration step, resulting in the first linear-time algorithm for solving CVaR MDPs with bounded error.

Lemma 1. *Any discrete distribution has a piecewise linear and convex $yCVaR_y$ function. Similarly, any piecewise linear convex function can be seen as representing a certain discrete distribution.*
Particularly, the integral of the quantile function is the $yCVaR_y$ function

$$yCVaR_y(Z) = \int_0^y VaR_\beta(Z)\,d\beta \tag{3.9}$$

and the derivative of the $yCVaR_y$ function is the quantile function

$$\frac{\partial}{\partial y} yCVaR_y(Z) = VaR_y(Z) \tag{3.10}$$

Proof. The fact that discrete distributions have a piecewise linear $y\text{CVaR}_y$ function has already been shown by Rockafellar and Uryasev [47].
According to definition (2.16) we have

$$y\text{CVaR}_y(Z) = y\frac{1}{y}\int_0^y \text{VaR}_\beta(Z)\text{d}\beta = \int_0^y \text{VaR}_\beta(Z)\text{d}\beta$$

by taking the y derivative, we have

$$\frac{\partial}{\partial y} y\text{CVaR}_y(Z) = \frac{\partial}{\partial y}\int_0^y \text{VaR}_\beta(Z)d\beta = \text{VaR}_y(Z)$$

□

You can get some intuition from Figure 3.1, where the integral-derivation relationship is clearly visible. According to Lemma 1, we can reconstruct the $y\text{CVaR}_y$ from the underlying distribution and vice-versa. We utilize the fact that the conversion is linear in the number of probability atoms to formulate a fast way of computing the $\mathbf{T}_\mathcal{I}$ operator.

3.3.1 *CVaR Computation via Quantile Representation*

We propose the following procedure: instead of using linear programming for the CVaR computation, we use the underlying distributions represented by the $y\text{CVaR}_y$ function to compute CVaR at each atom. The general steps of the computation are:

1. transform $y\text{CVaR}_y(Z(x'))$ of each reachable state x' to a discrete probability distribution using (3.10).
2. combine these to to a distribution representing the full state-action distribution
3. compute $y\text{CVaR}_y$ for all atoms using (3.9)

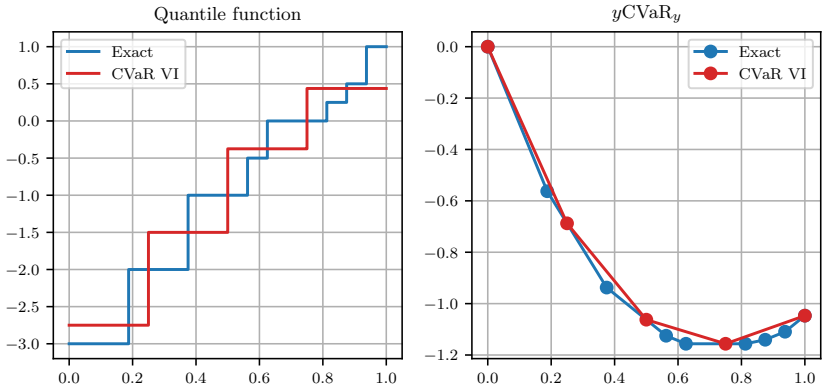


Figure 3.1: Comparison of a discrete distribution and its approximation according to the CVaR linear interpolation operator.

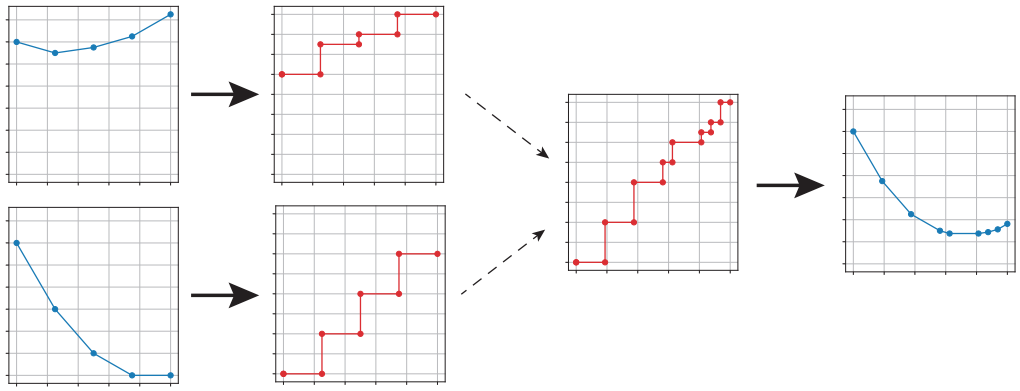


Figure 3.2: Visualization of the CVaR computation for a single state and action with two transition states. Thick arrows represent the conversion between $y\text{CVaR}_y$ and the quantile function.

See Figure 3.2 for a visualization of the procedure.
Note that this procedure is linear for discrete distributions. The only nonlinear step in the procedure is the sorting step in mixing distributions. Since the values are pre-sorted for each state x', this is equivalent to a single step of the Merge sort algorithm, which means it is also linear in the number of atoms.

We show the explicit computation of the procedure for linearly interpolated atoms in Algorithm 3.

The proposed method draws remarkable similarity to distributional RL (Section 2.2). In fact, if we overlook the action selection phase, the one-step update is identical. The main differences between the two approaches are 1) action selection - in the distributional approach, we select a single action (based on the expected value) for the whole distribution. In CVaR Value Iteration, we select a different action for each α-level; 2) the approximation step - in distributional RL, we try to minimize the Wasserstein distance of selected distributions. In CVaR Value Iteration, we construct a distribution whose CVaRs at given confidence levels are identical to the exact distribution.

To show the correctness of this approach, we formulate it as a solution to problem (3.2) in the next section. Note that we skip the reward and gamma scaling for readability's sake. Extension to the Bellman operator is trivial.

3.3.2 *ξ-computation*

Similarly to Theorem 2, we need a way to compute the $y_{t+1} = y_t \xi^*(x_t)$ to extract the optimal policy. We compute $\xi^*(x_t)$ by using the following intuition: y_{t+1} is the portion of $Z(x_{t+1})$ that is present in $\text{CVaR}_{y_t}(Z(x_t))$. In the continuous case, it is the probability in $Z(x_{t+1})$ before the $\text{VaR}_{y_t}(Z(x_t))$ as we show bellow.

Theorem 3. *Let x'_1, x'_2 be only two states reachable from state x via action a in a single transition. Let the cumulative distribution functions of the state's underlying distributions $Z(x'_1), Z(x'_2)$ be strictly increasing with unbounded support. Then the solution to minimization problem (3.2) can be computed by setting*

$$\xi(x'_i) = \frac{F_{Z(x'_i)}\left(F^{-1}_{Z(x,a)}(\alpha)\right)}{\alpha} \tag{3.11}$$

Proof. Since we are interested in the minimal argument, we can ease the computation by focusing on the αCVaR_α function instead of CVaR_α. When working with two states, the equation of interest simplifies to

$$\begin{aligned} \alpha\text{CVaR}_\alpha(Z(x,a)) = \min_\xi \; & p_1\xi_1\alpha\text{CVaR}_{\xi_1\alpha}\left(Z(x'_1)\right) + p_2\xi_2\alpha\text{CVaR}_{\xi_2\alpha}\left(Z(x'_2)\right) \\ \text{s.t.} \quad & p_1\xi_1 + p_2\xi_2 = 1 \\ & 0 \le \xi_1 \le \frac{1}{\alpha} \\ & 0 \le \xi_2 \le \frac{1}{\alpha} \end{aligned}$$

therefore

$$\begin{aligned} \alpha\text{CVaR}_\alpha(Z(x,a)) &= \min_\xi \; p_1\xi_1\alpha\text{CVaR}_{\xi_1\alpha}(Z(x'_1)) + (1-p_1)\frac{1-p_1\xi_1}{1-p_1}\alpha\text{CVaR}_{\frac{1-p_1\xi_1}{1-p_1}\alpha}(Z(x'_2)) \\ &= \min_\xi \; p_1\int_0^{\xi_1\alpha}\text{VaR}_\beta(Z(x'_1))\,\mathrm{d}\beta + (1-p_1)\int_0^{\frac{1-p_1\xi_1}{1-p_1}\alpha}\text{VaR}_\beta(Z(x'_2)\,\mathrm{d}\beta \end{aligned}$$

To find the minimal argument, we find the first derivative w.r.t. ξ_1

$$\begin{aligned} \frac{\partial\alpha\text{CVaR}_\alpha}{\partial\xi_1} &= p_1\alpha\text{VaR}_{\xi_1\alpha}(Z(x'_1)) + (1-p_1)\alpha\frac{-p_1}{1-p_1}\text{VaR}_{\frac{1-p_1\xi_1}{1-p_1}\alpha}(Z(x'_2) \\ &= p_1\text{VaR}_{\xi\alpha}(Z(x'_1)) - p_1\text{VaR}_{\frac{1-p\xi}{1-p}\alpha}(Z(x'_2)) \end{aligned}$$

By setting the derivative to 0 , we get

$$\text{VaR}_{\xi_1\alpha}(Z(x'_1)) \overset{!}{=} \text{VaR}_{\frac{1-p\xi}{1-p}\alpha}(Z(x'_2)) = \text{VaR}_{\xi_2\alpha}(Z(x'_2))$$

Bernard and Vanduffel [13] have shown that in the case of strictly increasing c.d.f. with unbounded support, it holds that

$$\begin{aligned} \text{VaR}_{\xi_1\alpha}(Z(x'_1)) = \text{VaR}_{\xi_2\alpha}(Z(x'_2)) \quad &= \text{VaR}_\alpha(Z(x,a)) \\ F^{-1}_{Z(x'_1)}(\xi_1\alpha) = F^{-1}_{Z(x'_2)}(\xi_2\alpha) \quad &= F^{-1}_{Z(x,a)}(\alpha) \end{aligned}$$

and we can extract the values of $\xi_1\alpha, \xi_2\alpha$ using the

$$\begin{aligned} F^{-1}_{Z(x'_1)}(\xi_1\alpha) &= F^{-1}_{Z(x,a)}(\alpha) && / F_{Z(x'_1)} \\ F_{Z(x'_1)}\left(F^{-1}_{Z(x'_1)}(\xi_1\alpha)\right) &= F_{Z(x'_1)}\left(F^{-1}_{Z(x,a)}(\alpha)\right) \\ \xi_1\alpha &= F_{Z(x'_1)}\left(F^{-1}_{Z(x,a)}(\alpha)\right) \end{aligned}$$

And similarly for ξ_2.

Since the problem is convex, we have found the optimal point.

□

The theorem is straightforwardly extendable to multiple states by induction. We conjecture that similar claim holds for general distributions, however this would require more technical arguments and is out of scope of this thesis. Among other difficulties, the optimal ξ is not unique for general distributions. See Bernard and Vanduffel [13] for details on the two-dimensional case for general distributions.

3.4 EXPERIMENTS

We test the proposed algorithm on the same task as Chow et al. [20]. The task of the agent is to navigate on a rectangular grid to a given destination, moving in its four-neighborhood. To encourage fast movement towards the goal, the agent is penalized for each step by receiving a reward -1. A set of obstacles is placed randomly on the grid and stepping on an obstacle ends the episode while the agent receives a reward of -40. To simulate sensing and control noise, the agent has a $\delta = 0.05$ probability of moving to a different state than intended.

For our experiments, we choose a 40×60 grid-world and approximate the αCVaR_α function using 21 log-spaced atoms with $\theta = 2$ as in [20]. The learned policies on a sample grid are shown in Figure 3.3.

While Chow et al. [20] report computation time on the order of two hours, our naive Python implementation converged within 20 minutes and there is ample room for improvement.

3.4.1 *Empirical difficulties*

We have encountered slight difficulties when testing the algorithms. While the CVaR value estimates monotonically converged towards the contraction fixed points, the same cannot be said about the extracted policies. Some policies failed to reach the goal via the optimal paths and some even got stuck in cycles.

After some investigation, we identified that these nonoptimal behaviors happen around points where a small $\alpha-$change affects the selected actions and are caused by the approximation errors around those points. We note that similar problems also affect the distributional RL approaches, as one can find examples of distributions whose expected value differs dramatically from the value approximated by the Wasserstein optimum.

These problems can always resolved by increasing the number of atoms around these critical points, as both methods are consistent in the limit.

Algorithm 3 CVaR Computation via Quantile Representation

function extractDistribution
  input: vectors $\mathbf{C}, \mathbf{y}$
  # Note: $y_0 = C(x', y_0) = 0$
  for $i \in \{1, ..., |\mathbf{y}|\}$ **do**
    $d_i = \dfrac{C(x', y_i) - C(x', y_{i-1})}{y_i - y_{i-1}}$
  end for
  output vector $\mathbf{d}$

function extractC
  input: vectors $\mathbf{d}, \mathbf{p}$
  $C_0 = 0$
  for $i \in \{1, ..., |\mathbf{p}|\}$ **do**
    $C_i = C_{i-1} + d_i \cdot p_i$
  end for
  output vector $\mathbf{C}$

function mixDistributions
  input: tuples $(\mathbf{d^{(1)}}, p^{(1)}), ..., (\mathbf{d^{(K)}}, p^{(K)})$ and vector $\mathbf{y}$
  # $\sum_{k=1}^{K} p_k = 1$
  for $i, k \in \{1, ..., K\} \times \{1, ..., |\mathbf{y}|\}$ **do**
    # Weigh atom probabilities by transitions
    $p_i^{(k)} = p^{(k)} \cdot (y_i - y_{i-1})$
  end for
  # Join all tuples together:
  $atoms = \left\{(d_1^{(1)}, p_1^{(1)}), ..., (d_N^{(1)}, p_N^{(1)}), (d_1^{(2)}, p_1^{(2)}), ..., (d_N^{(K)}, p_N^{(K)})\right\}$
  Sort *atoms* by d
  Unwrap vectors $\mathbf{d}, \mathbf{p}$ from sorted tuples
  output $\mathbf{d}, \mathbf{p}$

Main
input: tuples $(C(x_i', \bullet), p^{(1)}), ..., (C(x_i', \bullet), p^{(K)})$ and vector $\mathbf{y}$
for $i \in \{1, ..., K\}$ **do**
  $\mathbf{d^{(i)}}$ = extractDistribution$(C(x_i', \bullet), \mathbf{y})$
end for
$\mathbf{d_{mix}}, \mathbf{y_{mix}}$ = mixDistributions$((\mathbf{d^{(1)}}, p^{(1)}), ..., (\mathbf{d^{(K)}}, p^{(K)}), \mathbf{y})$
$\mathbf{C_{out}}$ = extractC$(\mathbf{d_{mix}}, \mathbf{y_{mix}})$
output: $\mathbf{C_{out}}$

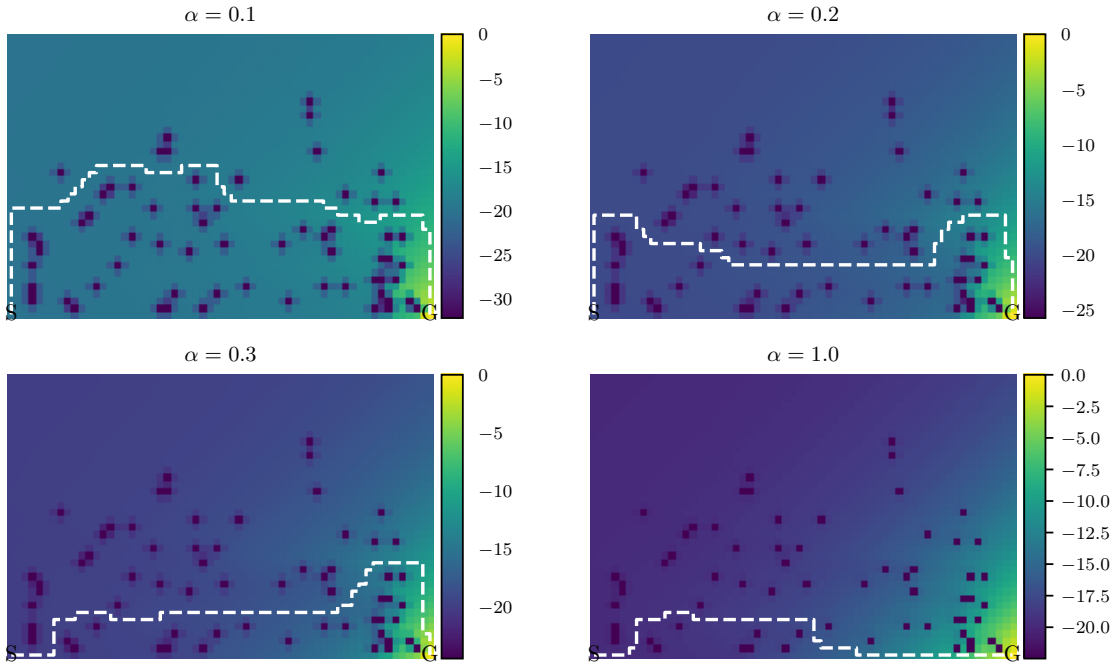


Figure 3.3: Grid-world simulations. The optimal deterministic paths are shown together with CVaR estimates for given α.

4

Q-LEARNING WITH CVAR

While value iteration is a useful algorithm, it only works when we have complete knowledge of the environment - including the probability transitions $p(x'|x,a)$. This is often not the case in practice and we have to rely on different methods, based on direct interaction with the environment. One such algorithm is the well-known Q-learning which we explore in this chapter.

We first remind the reader of Q-learning basics in Section 4.1 and introduce CVaR estimation in Section 4.2. These concepts are combined together with CVaR value iteration and in Section 4.3 we propose the new CVaR Q-learning algorithm. We treat the optimal policy separately in Section 4.4.

The algorithm is then experimentally verified on suitable environments in Section 4.5.

4.1 Q-LEARNING

Q-learning (Watkins and Dayan [59]) is an important off-policy Temporal Difference control algorithm, that works by repeatedly updating the Q value estimate according to the sampled rewards and states using a moving exponential average.

$$\begin{aligned} &Q_{t+1}(x,a) = (1-\beta_t)Q_t(x,a) + \beta_t \left[r + \gamma \max_{a'} Q_t(x',a') \right] \\ &x' \sim p(\cdot|x,a) \end{aligned} \tag{4.1}$$

Here Q is an estimate of the optimal action-value function (2.7) and β_t is the learning rate at time t. The expression $r + \gamma \max_{a'} Q_t(x',a')$ is called a *target* and is sometimes denoted as $\mathcal{T}Q$. The idea of the algorithm is to bring the value function closer to the target, which is more 'informed' than the original value since it has information about the reward and next state, which came directly from the sampled transition. The optimal value of $Q(x,a)$ is then the expected target, which we reach in the limit.

The order of the visited states is unimportant, as long as all reachable states are updated infinitely often and the learning rate meets a standard condition used in stochastic approximation.

$$\sum_{t=0}^{\infty} \beta_t = \infty \quad \sum_{t=0}^{\infty} \beta_t^2 < \infty \tag{4.2}$$

See Jaakkola, Jordan, and Singh [29] for details.

While the algorithm would converge if we were using a completely random policy, in practice we often try to speed up the convergence by using a smarter, yet still random policy as seen in Algorithm 4.

Algorithm 4 Q-learning

Initialize $Q(x, a)$ for all $x \in \mathcal{X}, a \in \mathcal{A}$ arbitrarily, and $Q(x_{\text{terminal}}, \cdot) = 0$
for each episode **do**
  $x = x_0$
  while x is not terminal **do**
    Choose a using a policy derived from Q (e.g. ε-greedy)
    Take action a, observe r, x'
    $Q(x, a) = (1 - \beta_t)Q(x, a) + \beta_t \left[r + \gamma \max_{a'} Q(x', a')\right]$
    $x = x'$
  end while
end for

4.2 CVAR ESTIMATION

Before formulating a CVaR version of Q-learning, we must first talk about simply *estimating* CVaR, as it is not as straightforward as the estimation of expected value.

Let us remind ourselves of the primal definition of CVaR (2.17):

$$\text{CVaR}_\alpha(Z) = \max_s \left\{ \frac{1}{\alpha} \mathbb{E}\left[(Z - s)^-\right] + s \right\}$$

If we knew the exact $s^* = \text{VaR}_\alpha$, we could estimate the CVaR as a simple expectation of the $\frac{1}{\alpha}(Z - s^*)^- + s^*$ function. As we do not know this value in advance, a common approach is to first approximate VaR_α from data, then use this estimate to compute its CVaR_α. This is usually done with a full data vector, requiring the whole data history to be saved in memory.

When dealing with reinforcement learning, we would like to store our current estimate as a scalar instead. This requires finding a recursive expression whose expectation is the CVaR value. Fortunately, similar methods have been thoroughly investigated in the stochastic approximation literature by Robbins and Monro [46].

The RM theorem has also been applied directly to CVaR estimation by Bardou, Frikha, and Pages [7], who used it to formulate a recursive importance sampling procedure useful for estimating CVaR of long-tailed distributions.

First let us describe the method for one step estimation, meaning we sample values (or rewards in our case) r from some distribution and our goal is to estimate CVaR at a given confidence level α. The procedure requires us to maintain two separate estimates V and C, being our VaR and CVaR estimates respectively.

$$V_{t+1} = V_t + \beta_t \left[1 - \frac{1}{\alpha} \mathbb{1}_{(V_t \geq r)}\right] \tag{4.3}$$

$$C_{t+1} = (1 - \beta_t)C_t + \beta_t \left[V_t + \frac{1}{\alpha}(r - V_t)^-\right] \tag{4.4}$$

An observant reader may recognize a standard equation for quantile estimation in equation (4.3) (see e.g. Koenker and Hallock [32] for more information on quantile estimation/regression). The expectation of the update $\mathbb{E}\left[1 - \frac{1}{\alpha} \mathbb{1}_{(V_t \geq r)}\right]$ is the inverse gradient of the CVaR primal definition, so we are in fact performing a Stochastic Gradient Descent on the primal.

Equation (4.4) is also quite intuitive, representing the moving exponential average of the primal CVaR definition (2.17). The estimations are proven to converge, given the usual requirements on the learning rate (4.2) [7].

4.3 CVAR Q-LEARNING

We now extend the previously established CVaR Value Iteration and combine it with the recursive CVaR estimation techniques to formulate a new algorithm we call CVaR Q-learning.

4.3.1 *Temporal Difference update*

We first define two separate values for each state, action, and atom $V, C : \mathcal{X} \times \mathcal{A} \times \mathcal{Y} \to \mathbb{R}$ where $C(x, a, y)$ represents $\text{CVaR}_y(Z(x, a))$ of the distribution, similar to the definition (3.3). $V(x, a, y)$ represents the VaR_y estimate, i.e. the estimate of the $y-$quantile of a distribution recovered from CVaR_y by Lemma 1.

A key to any temporal difference algorithm is its update rule. The CVaR TD update rule extends the improved value iteration procedure and we present the full rule for uniform atoms in Algorithm 5.

Let us now go through the algorithm step by step. We first construct a new CVaR (line 3), representing $\text{CVaR}_y(Z(x'))$, by greedily selecting actions that yield the highest CVaR for each atom. This is in contrast with both standard Q-learning and Quantile Regression Q-learning (Section 5.1.3) where we select a single action for the whole distribution. This step is implicit in the CVaR value iteration procedure since we are not working with action-value functions.

The new values $C(x', \cdot)$ are then transformed to the underlying distribution (line 5) $\mathbf{d}$ and used to create the target $\mathcal{T}\mathbf{d} = r + \gamma\mathbf{d}$. A natural Monte Carlo approach would be then to generate samples from this target distribution and use these to update our estimates V, C.

Since we know the target distributions exactly, we do not have to actually sample; instead we use the quantile values proportionally to their probabilities (in the uniform case, this means exactly once) and apply the respective VaR and CVaR update rules (lines 7, 8).

Algorithm 5 CVaR TD update

1: **input:** x, a, x', r
2: **for** each i **do**
3: $\quad C(x', y_i) = \max_{a'} C(x', a', y_i)$
4: **end for**
5: $\mathbf{d} = \text{extractDistribution}\,(C(x', \cdot), \mathbf{y})$ # See Algorithm 3
6: **for** each i, j **do**
7: $\quad V(x, a, y_i) = V(x, a, y_i) + \beta \left[1 - \frac{1}{y_i}\mathbb{1}_{(V(x,a,y_i) \geq r + \gamma d_j)}\right]$
8: $\quad C(x, a, y_i) = (1 - \beta)C(x, a, y_i) + \beta \left[V(x, a, y_i) + \frac{1}{y_i}\left(r + \gamma d_j - V(x, a, y_i)\right)^-\right]$
9: **end for**

If the atoms aren't uniformly spaced, we have to perform basic importance sampling when updating the estimates (nonuniform atoms are motivated by the error

bounds of CVaR Value Iteration). In contrast with the uniform version, we iterate only over the atoms and perform a single update for the whole target by taking an expectation over the target distribution. This is done by replacing lines 7, 8 with

$$\begin{aligned} V(x,a,y_i) &= V(x,a,y_i) + \beta \mathop{\mathbb{E}}_j \left[1 - \frac{1}{y_i}\mathbb{1}_{(V(x,a,y_i)\geq r+\gamma d_j)}\right] \\ C(x,a,y_i) &= (1-\beta)C(x,a,y_i) + \beta \mathop{\mathbb{E}}_j \left[V(x,a,y_i) + \frac{1}{y_i}\left(r+\gamma d_j - V(x,a,y_i)\right)^-\right] \end{aligned} \tag{4.5}$$

The explicit computation of the expectation term for VaR would then look like

$$\mathop{\mathbb{E}}_j \left[1 - \frac{1}{y_i}\mathbb{1}_{(V(x,a,y_i)\geq r+\gamma d_j)}\right] = \sum_j p_j \left[1 - \frac{1}{y_i}\mathbb{1}_{(V(x,a,y_i)\geq r+\gamma d_j)}\right]$$

where $p_j = y_j - y_{j-1}$ represents the probability of d_j. The CVaR update expectation is computed analogically.

This is a valid approach since sample mean is equal to the mean of the original distribution. In this case we are performing the updates on batches of samples and the final expected value remains unchanged.

$$\mathbb{E}[f(Z)] = \sum_i p_i \mathbb{E}[f(Z_i)]$$

The above equation holds for any function f if Z is a mixture of Z_i, so it also holds for the VaR update $1 - \frac{1}{y_i}\mathbb{1}_{(V(x,a,y_i)\geq r+\gamma d_j)}$ where the learned distribution is a mixture of the different target distributions.

We conclude the same for the CVaR update, since the expectation remains unchanged.

We are in fact using more informed updates - similar to the difference between pure and batch Stochastic Gradient Descent.

4.3.2 *Note on convergence*

We do not prove the convergence of the CVaR Q-learning algorithm in this thesis as it would require significant work regarding convergence of the recursive CVaR estimation procedures. The update rules (4.3, 4.4) have been only shown to converge if the underlying distributions are continuous [7], which is not the case in our setting.

In the last section of this chapter, we show at least empirical convergence of the CVaR Q-learning algorithm.

4.4 OPTIMAL POLICY

Recall that in CVaR Value Iteration we can extract the optimal policy by recursively setting $y_{t+1} = y_t \xi^*_{x_{t+1}}$. This process is not straightforwardly extendable to our sample-based version of CVaR Value Iteration, since we would have to have access to all possible transition states and probabilities in order to compute ξ^*.

Instead, we turn to the primal formulation of CVaR in what we call VaR-based policy improvement algorithm. We first introduce the VaR-based policy improvement

in the context of distributional RL and prove its validity. The policy improvement procedure is then used as a consistent heuristic for extracting the optimal policy from converged CVaR Value Iteration.

4.4.1 *VaR-based Policy Improvement*

Let us now assume that we have successfully converged with distributional value iteration and have available the return distributions of some stationary policy for each state and action. Our next goal is to find a policy improvement algorithm that will monotonically increase the CVaR_α criterion for selected α.

Recall the primal definition of CVaR (2.17)

$$\text{CVaR}_\alpha(Z) = \max_s \left\{ \frac{1}{\alpha} \mathbb{E}\left[(Z-s)^-\right] + s \right\}$$

Our goal (2.21) can then be rewritten as

$$\max_\pi \text{CVaR}_\alpha(Z^\pi) = \max_\pi \max_s \frac{1}{\alpha} \mathbb{E}\left[(Z^\pi - s)^-\right] + s$$

As mentioned earlier, the primal solution is equivalent to $\text{VaR}_\alpha(Z)$

$$\text{CVaR}_\alpha(Z) = \max_s \left\{ \frac{1}{\alpha} \mathbb{E}\left[(Z-s)^-\right] + s \right\} = \frac{1}{\alpha} \mathbb{E}\left[(Z - \text{VaR}_\alpha(Z))^-\right] + \text{VaR}_\alpha(Z)$$

The main idea of VaR-based policy improvement is the following: If we knew the value s^* in advance, we could simplify the problem to maximize only

$$\max_\pi \text{CVaR}_\alpha(Z^\pi) = \max_\pi \frac{1}{\alpha} \mathbb{E}\left[(Z^\pi - s^*)^-\right] + s^* \tag{4.6}$$

Given that we have access to the return distributions, we can improve the policy by simply choosing an action that maximizes CVaR_α in the first state $a_0 = \arg\max_\pi \text{CVaR}_\alpha(Z^\pi(x_0))$, setting $s^* = \text{VaR}_\alpha(Z(x_0, a_0))$ and focus on maximization of the simpler criterion.

This can be seen as coordinate ascent with the following phases:

1. Maximize $\frac{1}{\alpha}\mathbb{E}\left[(Z^\pi(x_0) - s)^-\right] + s$ w.r.t. s while keeping π fixed. This is equivalent to computing CVaR according to the primal.
2. Maximize $\frac{1}{\alpha}\mathbb{E}\left[(Z^\pi(x_0) - s)^-\right] + s$ w.r.t. π while keeping s fixed. This is the policy improvement step.
3. Recompute $\text{CVaR}_\alpha(Z^{\pi^*})$ where π^* is the new policy.

Since our goal is to optimize the criterion of the distribution starting at x_0, we need to change the value s while traversing the MDP (where we have only access to $Z(x_t)$). We do this by recursively updating the s we maximize by setting $s_{t+1} = \frac{s_t - r}{\gamma}$. See Algorithm 6 for the full algorithm which we justify in the following theorem.

Theorem 4. *Let π be a stationary policy, $\alpha \in (0, 1]$. By following policy π^* from algorithm 6, we improve $CVaR_\alpha(Z)$ in expectation:*

$$CVaR_\alpha(Z^\pi) \leq CVaR_\alpha(Z^{\pi^*})$$

Algorithm 6 VaR-based policy improvement

$a = \arg\max_a \mathrm{CVaR}_\alpha(Z(x_0, a))$
$s = \mathrm{VaR}_\alpha(Z(x_0, a))$
Take action a, observe x, r
while x is not terminal **do**
  $s = \dfrac{s - r}{\gamma}$
  $a = \arg\max_a \mathbb{E}\left[(Z(x, a) - s)^-\right]$
  Take action a, observe x, r
end while

Proof. Let s^* be a solution to $\max_s \frac{1}{\alpha}\mathbb{E}\left[(Z^\pi(x_0) - s)^-\right] + s$. Then by optimizing $\frac{1}{\alpha}\mathbb{E}\left[(Z^\pi - s^*)^-\right]$ over π, we monotonously improve the optimization criterion $CVaR_\alpha(Z(x_0))$.

$$\begin{aligned}
\mathrm{CVaR}_\alpha(Z^\pi) = \max_s \frac{1}{\alpha}\mathbb{E}\left[(Z^\pi - s)^-\right] + s \qquad &= \frac{1}{\alpha}\mathbb{E}\left[(Z^\pi - s^*)^-\right] + s^* \\
\leq \max_{\pi'} \frac{1}{\alpha}\mathbb{E}\left[(Z^{\pi'} - s^*)^-\right] + s^* \qquad &= \frac{1}{\alpha}\mathbb{E}\left[(Z^{\pi^*} - s^*)^-\right] + s^* \\
\leq \max_{s'} \frac{1}{\alpha}\mathbb{E}\left[(Z^{\pi^*} - s')^-\right] + s' \qquad &= CVaR_\alpha(Z^{\pi^*})
\end{aligned}$$

When optimizing w.r.t. π we can ignore the scaling term $\frac{1}{\alpha}$ and a constant term s^* without affecting the optimal argument. We can therefore focus on optimization of $\mathbb{E}\left[(Z^\pi(x_0) - s^*)^-\right]$.

$$\begin{aligned}
\mathbb{E}\left[(Z_t - s)^-\right] &= \mathbb{E}\left[(Z_t - s)\mathbb{1}(Z_t \leq s)\right] = \mathbb{E}\left[(r_t + \gamma Z_{t+1} - s)\mathbb{1}(Z_{t+1} \leq \frac{s - r_t}{\gamma})\right] \\
&= \sum_{x_{t+1}, r_t} P(x_{t+1}, r_t \mid x_t, a)\mathbb{E}\left[(r_t + \gamma Z(x_{t+1}) - s)\mathbb{1}(Z(x_{t+1}) \leq \frac{s - r_t}{\gamma})\right] \\
&= \sum_{x_{t+1}, r_t} P(x_{t+1}, r_t \mid x_t, a)\mathbb{E}\left[\gamma\left(Z(x_{t+1}) - \frac{s - r_t}{\gamma}\right)\mathbb{1}(Z(x_{t+1}) \leq \frac{s - r_t}{\gamma})\right] \\
&= \gamma \sum_{x_{t+1}, r_t} P(x_{t+1}, r_t \mid x_t, a)\mathbb{E}\left[\left(Z(x_{t+1}) - \frac{s - r_t}{\gamma}\right)\mathbb{1}(Z(x_{t+1}) \leq \frac{s - r_t}{\gamma})\right] \\
&= \gamma \sum_{x_{t+1}, r_t} P(x_{t+1}, r_t \mid x_t, a)\mathbb{E}\left[\left(Z(x_{t+1}) - \frac{s - r_t}{\gamma}\right)^-\right]
\end{aligned} \tag{4.7}$$

where we used the definition of return $Z_t = R_t + \gamma Z_{t+1}$ and the fact that probability mixture expectations can be computed as $\mathbb{E}[f(Z)] = \sum_i p_i \mathbb{E}[f(Z_i)]$ for any function f.

Now let's say we sampled reward r_t and state x_{t+1}, we are still trying to find a policy π^* that maximizes

$$\begin{aligned}
\pi^* &= \arg\max_\pi \mathbb{E}\left[(Z(x_t) - s)^- \,\middle|\, x_{t+1}, r_t\right] \\
&= \arg\max_\pi \mathbb{E}\left[\left(Z(x_{t+1}) - \frac{s - r_t}{\gamma}\right)^-\right]
\end{aligned} \tag{4.8}$$

Where we ignored the unsampled states, since these are not a function of x_{t+1}, and the multiplicative constant γ that will not affect the maximum argument.

At the starting state, we set $s = s^*$. At each following state we select an action according to equation (4.8). By induction we maximize the criterion (4.6) in each step. □

Note that while the resulting policy is nonstationary, we do not need an extended state-space to follow this policy. It is only necessary to remember our previous value of s.

The ideas presented here were partially explored by Bäuerle and Ott [8] although not to this extent. See Remark 3.9 in [8] for details.

4.4.2 *CVaR Q-learning extension*

We would now like to use the policy improvement algorithm in order to extract the optimal policy from CVaR Q-learning. This would mean optimizing $\mathbb{E}\left[(Z_t - s)^-\right]$ in each step. A problem we encounter here is that we have access only to the discretized distributions and we cannot extract the values between selected atoms.

As a solution to this, we propose an approximate heuristic that uses linear interpolation to extract the VaR of given distribution.

The expression $\mathbb{E}\left[(Z_t - s)^-\right]$ is computed by taking the expectation of the distribution *before* the value s. We are therefore looking for value y where $\mathrm{VaR}_y = s$. This value is linearly interpolated from $\mathrm{VaR}_{y_{i-1}}$ and VaR_{y_i} where $y_i = \min\{y : \mathrm{VaR}_y \geq s\}$. The expectation is then taken over the extracted distribution, as this is the distribution that approximates CVaR the best.

See Algorithm 7 for the exact procedure and Figure 4.1 for more intuition behind the heuristic.

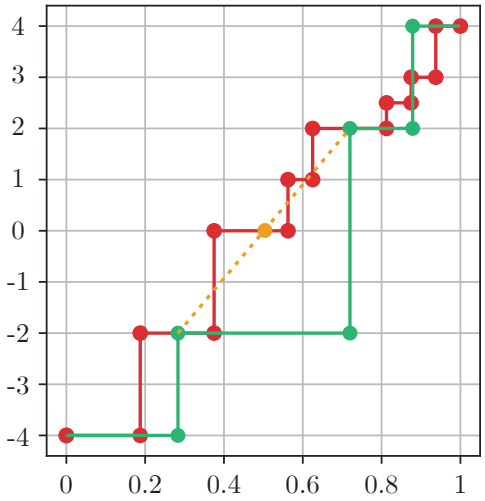


Figure 4.1: Visualization of the VaR-based heuristic. Quantile function of the exact distribution (unknown to the model) is shown in red and the VaR estimates at selected α-levels are shown in green. Let's say we now want to know y where $\mathrm{VaR}_y = 0$. We use linear interpolation between the nearest known VaRs, shown in orange. In this case the interpolation estimate is $y = 0.5$.

Algorithm 7 CVaR Q-learning policy

input: α, converged V, C
$x = x_0$
$a = \arg\max_a C(x, a, \alpha)$
$s = V(x, a, y)$
while x is not terminal **do**
  $\mathbf{d}_a = \text{extractDistribution}\,(C(x', a, \bullet), \mathbf{y})$ for each a
  $a = \arg\max_a \text{expMinInterp}(s, \mathbf{d}, V(x', a, \bullet), \mathbf{y})$
  Take action a, observe r, x'
  $s = \dfrac{s - r}{\gamma}$
  $x = x'$
end while

Compute $\mathbb{E}\left[(\mathbf{d}_a - s)^-\right]$ with linear interpolation
function expMinInterp
  input: s, vectors $\mathbf{d}, \mathbf{V}, \mathbf{y}$
  $z = 0$
  for $i \in \{1, ..., |\mathbf{y}|\}$ **do**
    if $S < V_i$ **then**
      break
    end if
    $z = z + d_i \cdot (y_i - y_{i-1})$
  end for
  $p_{\text{last}} = \dfrac{s - V_{i-1}}{V_i - V_{i-1}}(y_i - y_{i-1})$
  $z = z + d_i \cdot p_{\text{last}}$
  output z

4.5 EXPERIMENTS

We use the same gridworld from Section 3.4 with $\delta = 0.1$. Since the positive reward is very sparse, we chose to run CVaR Q-learning on a smaller environment of size 10×15. We trained the agent for 10,000 sampled episodes with learning rate $\beta = 0.4$ that dropped each 10 episodes by a factor of 0.995. The used policy was $\varepsilon-$greedy and maximized expected value ($\alpha = 1$) with $\varepsilon = 0.5$. Notice the high value of ε. We found that lower ε values led to overfitting the optimal expected value policy as the agent updated states out of the optimal path sparsly. This point will be elaborated further in the experimental section of the next chapter.

With said parameters, the agent was able to learn the optimal policies for different levels of α. See Figure 4.2 for learned policies and Figure 4.3 for Monte Carlo comparisons.

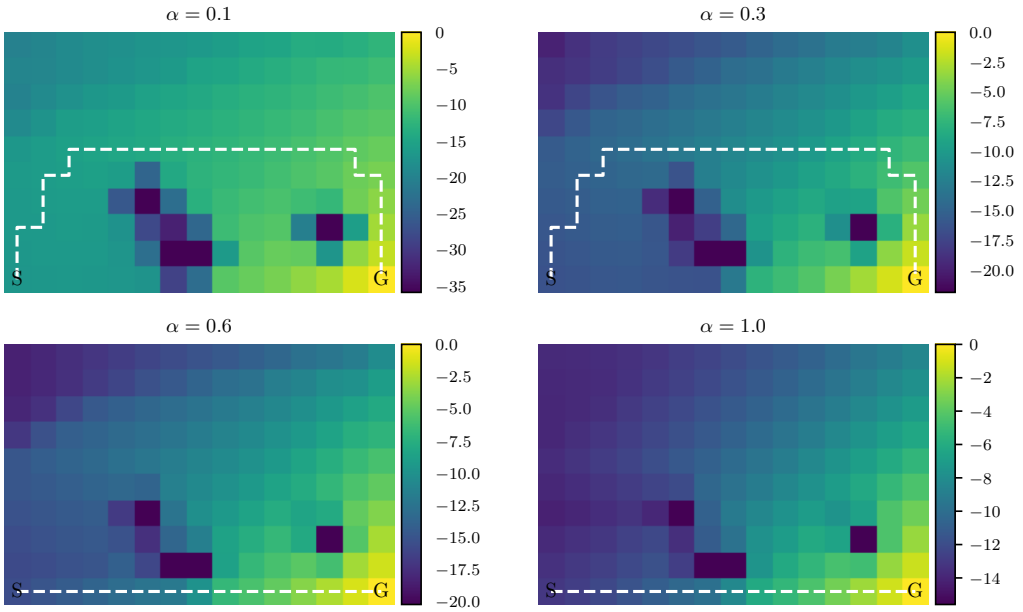


Figure 4.2: Grid-world Q-learning simulations. The optimal deterministic paths are shown together with CVaR estimates for given α.

The training was done with $N = 50$ linearly-spaced atoms. We experimented with several discretization settings and didn't find many differences between log- and linearly-spaced points. Note that learning VaR and CVaR for very low α values requires large number of samples and we found that extremely small atom values converged slowly.

Note on convexity: Unlike CVaR Value Iteration, where we maintain convexity of the $y\text{CVaR}_y$ function with each update (given we started with convex estimates), in CVaR Q-learning we can break the convexity in each update for any atom. We experience this in practice as well as can be gauged from Figure 4.4. Fortunately this fact does not break the update rule, since the targets we use to update C as well as V do not have to be ordered.

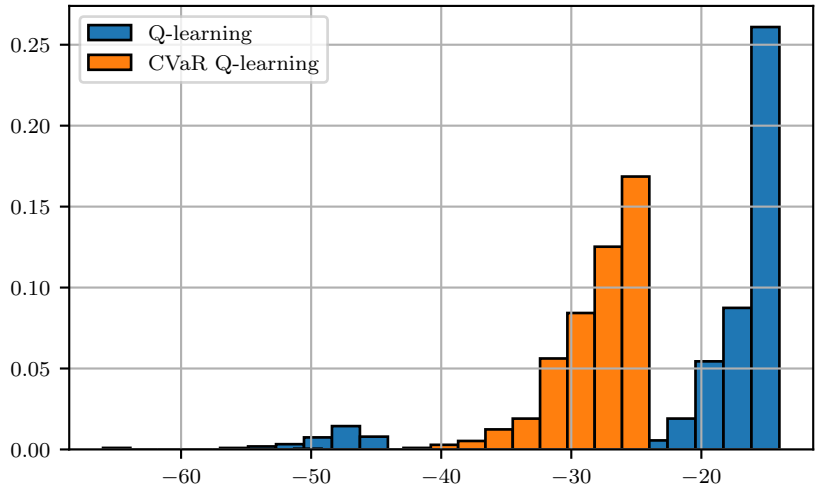


Figure 4.3: Histograms from 10000 runs generated by Q-learning and CVaR Q-learning with $\alpha = 0.1$.

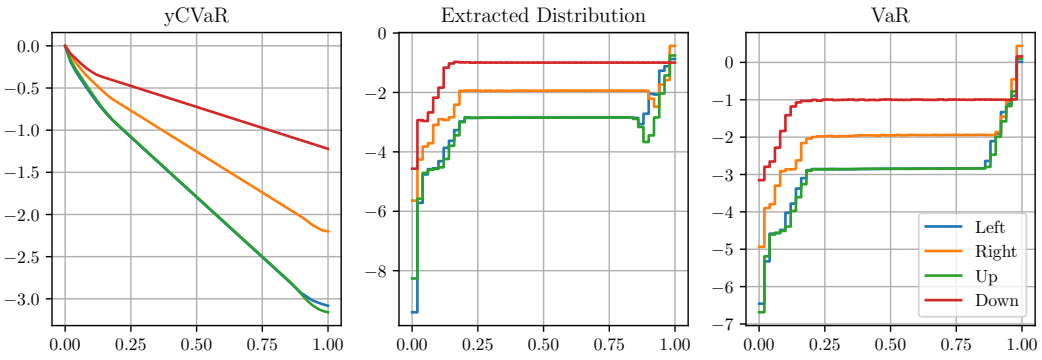


Figure 4.4: Learned C, V estimates for a single state after 10000 episodes. Notice the nonconvexities visible from the extracted distribution plot. Both extracted distribution and VaR functions should be nondecreasing.

5

DEEP Q-LEARNING WITH CVAR

A big disadvantage of value iteration and Q-learning is the necessity to store a separate value for each state. When the size of the state-space is too large, we are unable to store the action-value representation and the algorithms become intractable. To overcome this issue, it is common to use function approximation together with Q-learning. Mnih et al. [40] proposed the Deep Q-learning (DQN) algorithm and successfully trained on multiple different high-dimensional environments, resulting in the first artificial agent capable of learning a diverse array of challenging tasks.

In this chapter, we extend CVaR Q-learning to its deep Q-learning variant and show the practicality and scalability of the proposed methods.

5.1 DEEP Q-LEARNING

The ultimate goal of artificial intelligence are agents that exhibit a wide range of skills. In the past, research was often focused on narrow AI, which was able to perform well on one particular task, but was unable to generalize to other tasks well. This has changed with the advent of deep learning [35], that allowed us to train function approximators with the ability to auto-detect features. Deep learning methods have become popular across all of machine learning, especially for supervised learning and vision.

5.1.1 *Deep Learning*

We include a short glossary of deep learning basics and terminology. There are much better sources to draw from and we refer the reader to Goodfellow et al. [26] for a comprehensive overview of the field.

Deep learning considers a particular class of parametrized function approximators called *neural networks*. Neural networks are functions of the form

$$f_\theta(x) = f_n \circ f_{n-1} \circ \ldots \circ f_1(x)$$

where $f_i(x) = g_i(\theta_i x)$ with g_i being usually a nonlinear function called the *activation function* and θ_i is a parameter matrix. Every neural network has an input and output and several *layers*, where a layer represents one matrix multiplication and application of an activation function $f_i(x)$.

The types of neural networks differ, from the simplest with straightforward matrix multiplication called multi layered perceprtons (MLP) to more complicated ones. Arguably the most important layer type that kick-started the popularity of deep learning is called a *convolutional neural network* or CNN. Convolutional neural network

consists of several (even hundreds in recent vision applications) so called convolutional layers and its input is usually a 2D image. A convolutional layer consists of rectangular filters that look for increasingly abstract features by applying the same weight transformations over the whole image. The takeaway for us is that convolutional layers are able to learn from images, which is a fact we utilize in our experiments.

A necessary component of any deep learning algorithm is a loss function $\mathcal{L}$. The most common loss function being the mean squared error

$$\mathcal{L} = \mathbb{E}\left[(f_\theta(x) - \hat{y})^2\right]$$

with samples $\hat{y}$ representing the outputs drawn from a distribution of interest.

The loss function is then minimized using Stochastic Gradient Descent, a stochastic version of the well-known gradient descent (see e.g. Boyd and Vandenberghe [18]) where we perform each gradient step only using a subset of available samples. There exist many improved variants of SGD such as RMS-prop [57] or Adam [31], that perform significantly better than the vanilla SGD algorithm and are often used in practice. See Ruder [49] for a concise survey of the used algorithms.

5.1.2 *DQN*

Deep learning has also been succesfully applied to reinforcement learning. Q-learning has been used with function approximators in the past, however it suffers from instabilities during learning. In fact, it is well-known that Q-learning does not converge when used in conjunction with function approximators [6, 53] and this has been a problem in practice as well. Mnih et al. [40] were able to stabilize the learning process by introducing two practical techniques. Firstly the model isn't learned online, meaning we see each example once, but instead a replay buffer is used to store transitions (x, a, r, x') and these are later randomly sampled and used repeatedly for the updates. Secondly, a second network Q' is used for computing the target values $r + \gamma Q'(x', a')$ and is only slowly updated towards Q. These improvements have helped to stabilize the learning greatly. See Algorithm 8 for the full procedure.

The DQN algorithm has been applied on the Atari Learning Environment [9], a set of challenging and diverse tasks, with both inputs and outputs mirroring a human's experience of playing and learning the Atari games, and the same algorithm achieved human and superhuman-level performance on many of these games.

5.1.3 *Distributional Reinforcement Learning with Quantile Regression*

Before we transition to CVaR Q-learning, we will mention the Quantile Regression DQN algorithm by Dabney et al. [23], which shares certain similarities with CVaR Q-learning.

In QR-DQN, the goal is to learn a distribution that minimizes the Wasserstein distance from the actual return distribution, since the distributional value iteration operator is a contraction in Wasserstein distance.

The distributions are represented as N discrete uniform atoms with probability $\frac{1}{N}$ and are parametrized with a value that, when learned correctly, represents the $\frac{i+0.5}{N}$ quantile, which is the Wasserstein minimizer.

Algorithm 8 Deep Q-learning with experience replay

Initialize replay memory M
Initialize action-value function Q with random weights θ
Initialize target action-value function Q' with weights $\theta' = \theta$
for each episode **do**
  $x = x_0$
  while x is not terminal **do**
    Choose a using a policy derived from Q (ε-greedy)
    Take action a, observe r, x'
    Store transition (x, a, r, x') in M
    $x = x'$
    Sample random transitions (x_j, a_j, r_j, x'_j) from M
    Set $\mathcal{T}Q_j = r_j + \gamma \max_{a'} Q'(x'_j, a')$
    Perform a gradient step on $(\mathcal{T}Q_j - Q(x_j, a_j))^2$ w.r.t. θ
    Every N_{target} steps set $\theta' = \theta$
  end while
end for

The TD update rule for QR-DQN mimics the quantile estimation introduced in Chapter 4:

$$V_i(x,a) = V_i(x,a) + \beta \left[1 - \frac{1}{y_i}\mathbb{1}_{(V_i(x,a) \geq \mathcal{T}V_j)}\right]$$

where $\mathcal{T}V_j = r + \gamma V_j(x', a^*)$ and a^* is selected as the action that maximizes the expected value in the next state. The target is created for each atom y_j separately and differs from the one introduced in CVaR Q-learning.

The loss function reflects the asymmetry of the quantile and it is the standard quantile regression loss

$$\sum_{i=1}^{N} \mathop{\mathbb{E}}_{j} \left[(\mathcal{T}V_j - V_i(x,a))(y_j - \mathbb{1}_{(V_i(x,a) \geq \mathcal{T}V_j)})\right] \tag{5.1}$$

See Figure 5.1 for a visual comparison of loss functions used when finding expectations vs quantiles.

The algorithm is then extended to its deep Q-learning variant and verified empirically, which is a template we follow in the rest of this chapter.

5.2 DEEP CVAR Q-LEARNING

The transition from CVaR Q-learning to Deep CVaR Q-learning (CVaR DQN) follows the same principles as the one from Q-learning to DQN. Recall the TD update rule for CVaR Q-learning:

$$V(x,a,y_i) = V(x,a,y_i) + \beta \mathop{\mathbb{E}}_{j} \left[1 - \frac{1}{y_i}\mathbb{1}_{(V(x,a,y_i) \geq r + \gamma d_j)}\right]$$
$$C(x,a,y_i) = (1-\beta)C(x,a,y_i) + \beta \mathop{\mathbb{E}}_{j} \left[V(x,a,y_i) + \frac{1}{y_i}\left(r + \gamma d_j - V(x,a,y_i)\right)^-\right]$$

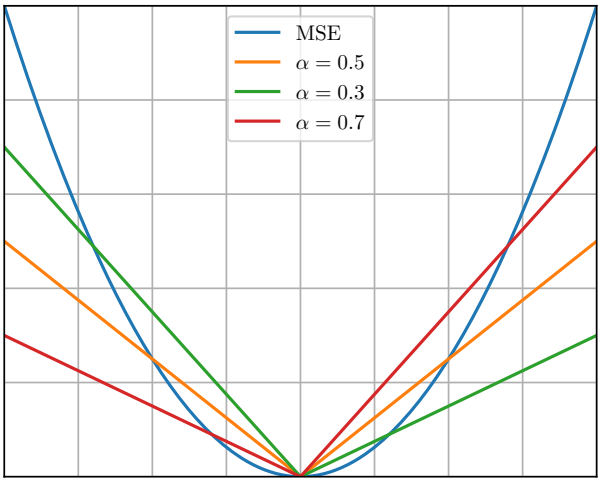


Figure 5.1: Comparison of quantile loss function and Mean Squared Error.

First significant change compared to DQN or QR-DQN is that we need to represent two separate values - one for V, one for C. As with DQN, we need to reformulate the updates as arguments minimizing some loss function.

5.2.1 *Loss functions*

The loss function for $V(x, a, y)$ is similar to QR-DQN loss in that we wish to find quantiles of a particular distribution. The target distribution however is constructed differently - in CVaR-DQN we extract the distribution from the $y\text{CVaR}_y$ function of the next state $\mathcal{T}V = r + \gamma\mathbf{d}$.

$$\mathcal{L}_{\text{VaR}} = \sum_{i=1}^{N} \mathop{\mathbb{E}}_{j} \left[(r + \gamma d_j - V_i(x, a))(y_j - \mathbb{1}_{(V_i(x,a) \geq r + \gamma d_j)}) \right] \tag{5.2}$$

where d_j are atoms of the extracted distribution.

Constructing the CVaR loss function consists of transforming the running mean into MSE, again with the transformed distribution atoms d_j

$$\mathcal{L}_{\text{CVaR}} = \sum_{i=1}^{N} \mathop{\mathbb{E}}_{j} \left[\left(V_i(x, a) + \frac{1}{y_i} (r + \gamma d_j - V_i(x, a))^{-} - C_i(x, a) \right)^2 \right] \tag{5.3}$$

Putting it all together, we are now able to construct the full CVaR-DQN loss function in Algorithm 9.

Combining the loss functions with the full DQN algorithm, we get the full CVaR-DQN with experience replay, see Algorithm 10. Note that we also utilize a target network C' that is used for extraction of the target values, similarly to the original DQN. The network V does not have a target network since the target is constructed independently of the value V.

Algorithm 9 Deep CVaR Loss function

input: x, a, x', r
  for each y_i **do**
    $C(x', y_i) = \max_{a'} C(x', a', y_i)$
  end for
  $\mathbf{d} = \text{extractDistribution}\left(C(x', \mathbf{y})\right)$
  $\mathcal{L}_{\text{VaR}} = \sum_{i=1}^{N} \mathbb{E}_j \left[(r + \gamma d_j - V_i(x,a))(y_j - \mathbb{1}_{(V_i(x,a) \geq r + \gamma d_j)})\right]$
  $\mathcal{L}_{\text{CVaR}} = \sum_{i=1}^{N} \mathbb{E}_j \left[\left(V_i(x,a) + \frac{1}{y_i}\left(r + \gamma d_j - V_i(x,a)\right)^- - C_i(x,a)\right)^2\right]$
return $\mathcal{L}_{\text{VaR}} + \mathcal{L}_{\text{CVaR}}$

Algorithm 10 Deep CVaR Q-learning with experience replay

Initialize replay memory M
Initialize the VaR function V with random weights θ_v
Initialize the CVaR function C with random weights θ_c
Initialize target CVaR function C' with weights $\theta'_c = \theta_c$
for each episode **do**
  $x = x_0$
  while x is not terminal **do**
    Choose a using a policy derived from C (ε-greedy)
    Take action a, observe r, x'
    Store transition (x, a, r, x') in M
    $x = x'$
    Sample random transitions (x_j, a_j, r_j, x'_j) from M
    Build the loss function $\mathcal{L}_{\text{VaR}} + \mathcal{L}_{\text{CVaR}}$ (Algorithm 9)
    Perform a gradient step on $\mathcal{L}_{\text{VaR}} + \mathcal{L}_{\text{CVaR}}$ w.r.t. θ_v, θ_c
    Every N_{target} steps set $\theta'_c = \theta_c$
  end while
end for

5.3 EXPERIMENTS

To test the approach in a complex setting, we applied the CVaR DQN algorithm to environments with visual state representation, which would be intractable for Q-learning without approximation.

5.3.1 *Atari*

We ran several experiments on the Arcade Learning Environment [9], which is used as a benchmark for many Deep Q-learning algorithms. The CVaR-DQN algorithm was able to learn reasonable policies with similar speed and performance as the original DQN algorithm. Unfortunately, due to the inherent determinism of the ALE, we didn't find significant differences between policies optimizing CVaR_α on different confidence levels, which led us to the construction of a new visual environment more suitable to risk-sensitive decision making.

5.3.2 *Ice Lake*

Ice Lake is a visual environment specifically designed for risk-sensitive decision making. Imagine you are standing on an ice lake and you want to travel fast to a point on the lake. Will you take the a shortcut and risk falling into the cold water or will you be more patient and go around? This is the basic premise of the Ice Lake environment which is visualized in Figure 5.2.

The agent has five discrete actions, namely go *Left, Right, Up, Down* and *Noop*. These correspond to moving in the respective directions or no operation. Since the agent is on ice, there is a sliding element in the movement - this is mainly done to introduce time dependency and makes the environment a little harder. The environment is updated thirty times per second.

The agent receives a negative reward of -1 per second, the episode ends with reward 100 if he reaches the goal unharmed or -50 if the ice breaks. This particular choice of reward leads to about a 15% chance of breaking the ice when taking the shortcut and it is still advantageous for a risk-neutral agent to take the dangerous path.

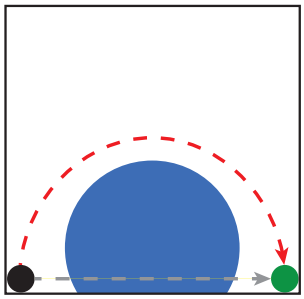


Figure 5.2: The Ice Lake environment. The agent is black and his target is green. The blue ring represents a dangerous area with risk of breaking the ice. Grey arrow shows the optimal risk-neutral path, red shows the risk-averse path.

5.3.3 *Network Architecture*

During our experiments we used a simple Multi-Layered Perceptron with 64 hidden units for a baseline experiment and later the original DQN architecture with a visual representation. In our baseline experiments, the state was represented with x- y- position and velocity. After tuning training parameters, we switched to a visual representation of the state including the preprocessing tricks used in Mnih et al. [40] which we describe next.

Firstly, the input images are scaled to 84×84 pixels and converted to greyscale to ease memory requirements. In many visual environments, a single image cannot capture the full state information - for example detecting the velocity of different objects necessitates tracking the objects in time. This problem is solved by concatenating 4 subsequent images which are then used as a single state representation. The input for each state is then of size $84 \times 84 \times 4$.

The neural network used in DQN inputs the state representation and outputs a value for each discrete action. The first hidden layer convolves 32 filters of 8×8 with stride 4 with the input image and applies a rectifier nonlinearity activation function [30]. The second hidden layer convolves 64 filters of 4×4 with stride 2, again followed by a rectifier nonlinearity. This is followed by a third convolutional layer that convolves 64 filters of 3×3 with stride 1 followed by a rectifier. The final hidden layer is fully-connected and consists of 512 rectifier units. The output layer is a fully-connected linear layer with a single output for each valid action.

The architecture used in our experiments differs slightly from the original one used in DQN. In our case the output is not a single value but instead a vector of values for each action, representing CVaR_y or VaR_y for the different confidence levels y. This issue is reconciled by having the output of shape $|\mathcal{A}| \times N$ where N is the number of atoms we want to use and $|\mathcal{A}|$ is the action space size.

Another important difference is that we must work with two outputs - one for C, one for V. We have experimented with two separate networks (one for each value) and also with a single network differing only in the last layer. This approach may be advantageous, since we can imagine that the information required for outputting correct V or C is similar. Furthermore, having a single network instead of two eases the computation requirements.

We tested both approaches and since we didn't find significant performance differences, we settled on the faster version with shared weights. We also used 256 units instead of 512 to ease the computation requirements and used Adam [31] as the optimization algorithm.

The implementation was done in Python and the neural networks were built using Tensorflow [1] as the framework of choice for gradient descent. The code was based on OpenAi baselines [24], an open-source DQN implementation.

5.3.4 *Parameter Tuning*

During our experiments, we tested mostly with $\alpha = 1$ so as to find reasonable policies quickly. We noticed that the optimal policy with respect to expected value was found fast and other policies were quickly abandoned due to the character of ε-

greedy exploration. Unlike standard Reinforcement Learning, the CVaR optimization approach requires to find not one but in fact a continuous spectrum of policies - one for each possible α. This fact, together with the exploration-exploitation dilemma, contributes to the difficulty of learning the correct policies.

After some experimentation, we settled on the following points:

- The training benefits from a higher value of ε than DQN. While in DQN the final ε used during vast majority of the algorithm is 0.1, we settled on 0.3 as a reasonable value with the ability to explore faster, while making the learned trajectories exploitable.
- Training with a single policy is insufficient in larger environments. Instead of maximizing CVaR for $\alpha = 1$ as in our CVaR Q-learning experiments, we change the value α randomly for each episode (uniformly over $(0, 1]$).
- The random initialization used in deep learning has a detrimental effect on the initial distribution estimates, due to the way how the target is constructed and this sometimes leads to the introduction of extreme values during the initial training. We have found that clipping the gradient norm helps to mitigate these problems and overall helps with the stability of learning.

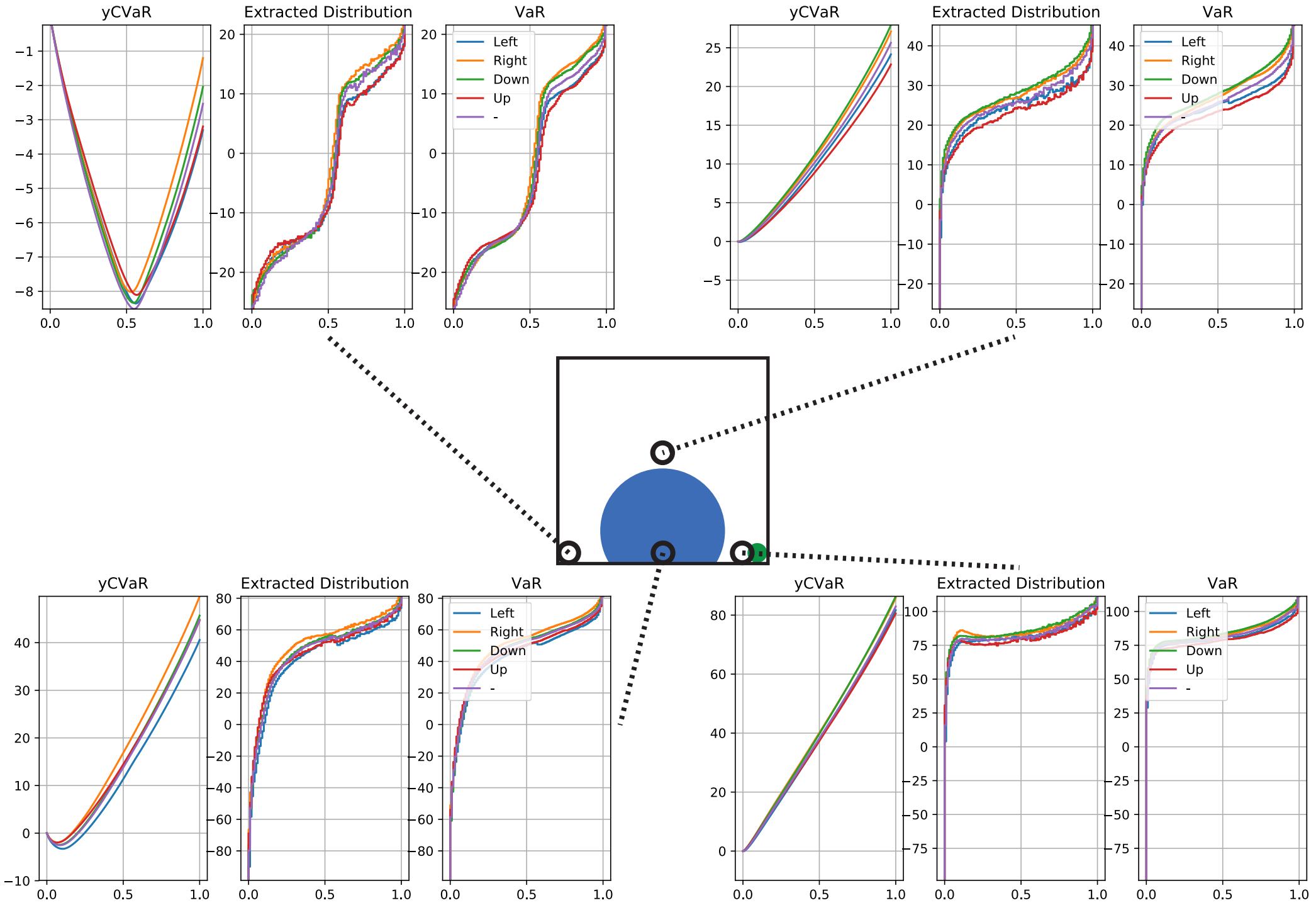
See table A.1 for a full list of hyperparameters and their values.

5.3.5 *Results*

With the tweaked parameters, both versions (baseline and visual) were able to converge and learned both the optimal expected value policy and the risk-sensitive policy. See Figure 5.3 on the next page for an overview of the final value function. **todo:** video

Although we tested with the vanilla version of DQN, we expect that all the DQN improvements such as experience replay [27], dueling [58], parameter noise [44] and others (combining the improvements matters, see [27]) should have a positive effect on the learning performance. Another practical improvement may be the introduction of Huber loss, similarly to QR-DQN.

Figure 5.3: Next page: CVaR DQN outputs for different positions in the Ice Lake environment.
In the starting state (top left), the model chooses *Right* for $\alpha > 0.5$ and *Up* for lower values, which is in accordance with a risk-sensitive agent. Near the target (bottom right), $y\text{CVaR}_y$ is close to linear - meaning the model is confident it will recieve a positive reward most of the time.


yCVaR
Extracted Distribution
VaR
Left
Right
Down
Up
-

6

CONCLUSION

In this thesis, we tackled the problem of dynamic risk-averse reinforcement learning. Specificaly we focused on optimizing the Conditional Value-at-Risk objective in a time-dependent Markov Decision Process, using methods of Reinforcement Learning.

The work mainly builds on the CVaR Value Iteration algorithm [20], a dynamic programming method for solving CVaR MDPs.

Our first original contribution is the proposal of a different computation procedure for CVaR value iteration. The novel procedure reduces the computation time from polynomial to linear. More specifically, our approach does not require solving a series of Linear Programs and instead finds solutions to the internal optimization problems in linear time by appealing to the underlying distributions of the CVaR function. We formally proved the equivalence of our solution for a subset of probability distributions.

Next we proposed a new sampling algorithm we call CVaR Q-learning, that builds on our previous results. Since the algorithm is sample-based, it does not require perfect knowledge of the environment. In addition, we proposed a new policy improvement algorithm for distributional reinforcement learning, proved its correctness and later used it as a heuristic for extracting the optimal policy from CVaR Q-learning. We empirically verified the practicality of the approach and our agent is able to learn multiple risk-sensitive policies all at once.

To show the scalability of the new algorithm, we extended CVaR Q-learning to its approximate variant by formulating the Deep CVaR loss function and used it in a deep learning context. The new Deep CVaR Q-learning algorithm with experience replay is able to learn different risk-sensitive policies from raw pixels.

We believe that the CVaR objective is a practical framework for computing control policies that are robust with respect to both stochasticity and model perturbations. Collectively, our work enhances the current state-of-the-art methods for CVaR MDPs and improves both practicality and scalability of the available approaches.

6.1 FUTURE WORK

Our contributions leave several pathways open for future work. Firstly, our proof of the improved CVaR Value Iteration works only for a subset of probability distributions and it shall be at least theoretically beneficial to prove the same for general distribution. The result may also be necessary for the convergence proof of CVaR Q-learning. Another missing piece required for proving the asymptotic convergence of CVaR Q-learning is the convergence of recursive CVaR estimation. Currently the

convergence has been proven only for continuous distributions and more general proof is required to show the CVaR Q-learning convergence.

In Chapter 4, we highlighted a way of extracting the current policy from converged CVaR Q-learning values. While the method is consistent in the limit, for practical purposes it serves only as a heuristic. It remains to be seen if there are better, perhaps exact ways of extracting the optimal policy.

The work of Bäuerle and Ott [8] shares a connection with CVaR Value Iteration and may be of practical use for CVaR MDPs. The relationship between CVaR Value Iteration and Bauerle's work is very similar to the c51 algorithm [10] and QR-DQN [23]. Bauerle's work is also a certain 'transposition' of CVaR Value Iteration and a comparison between the two may be beneficial. Of particular interest is the ease of extracting the optimal policy in a sampling version of the algorithm.

Lastly, our experimental work focused mostly on toy problems that demonstrated the basic functionality of the proposed algorithms. Since we believe our methods are practical beyond these toy settings, we would like to see the techniques applied on relevant problems from the financial sector and in the future on practical robotics, and other risk-sensitive applications.

A

APPENDIX

$$
\begin{aligned}
&\min_{\xi, I_{x'}} \frac{1}{y} \sum_{x'} p(x'|x,a) I_{x'} \\
&\text{s.t.} \quad 0 \leq \xi \leq \frac{1}{y} \\
&\qquad I_{x'} \geq y_i C(x, y_i) + \frac{y_{i+1} C(x, y_{i+1}) - y_i C(x, y_i)}{y_{i+1} - y_i} (y\xi(x') - y_i) \quad \forall i, \forall x' \\
&\qquad \sum_{x'} p(x'|x,a) \xi(x') = 1 \qquad \forall x'
\end{aligned}
\tag{A.1}
$$

Table A.1: CVaR DQN training parameters

Hyperparameter	**Value**	**Description**
minibatch size	32	Number of samples over which each SGD update is computed.
replay memory size	300000	SGD updates are sampled from this number of recent frames.
target network update frequency	5000	The frequency N_{target} with which the target network C' is updated.
γ	0.99	Discount Factor.
update frequency	4	We perform single gradient step each 4 frames.
learning rate	0.0001	We use Adam with $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon =$1e-8.
initial exploration	1.	Initial value of ϵ in ϵ-greedy.
final exploration	0.3	Initial value of ϵ in ϵ-greedy.
final exploration frame	1000000	ϵ is linearly annealed from initial to final value over this number of frames.
training starts	10000	First update is performed after what number of steps.
number of atoms	100	Number of uniformly spaced atoms to estimate quantiles.
gradient norm clipping	10	We clip the gradient if it's norm exceeds this value.

BIBLIOGRAPHY

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "TensorFlow: A System for Large-Scale Machine Learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[2] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. "Concrete problems in AI safety." In: *arXiv preprint arXiv:1606.06565* (2016).

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan." In: *arXiv preprint arXiv:1701.07875* (2017).

[4] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. "Coherent measures of risk." In: *Mathematical finance* 9.3 (1999), pp. 203–228.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." In: *arXiv preprint arXiv:1409.0473* (2014).

[6] Leemon Baird. "Residual algorithms: Reinforcement learning with function approximation." In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[7] Olivier Bardou, Noufel Frikha, and Gilles Pages. "Recursive computation of value-at-risk and conditional value-at-risk using MC and QMC." In: *Monte Carlo and quasi-Monte Carlo methods 2008*. Springer, 2009, pp. 193–208.

[8] Nicole Bäuerle and Jonathan Ott. "Markov decision processes with average-value-at-risk criteria." In: *Mathematical Methods of Operations Research* 74.3 (2011), pp. 361–379.

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. "The Arcade Learning Environment: An Evaluation Platform for General Agents." In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[10] Marc G Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning." In: *arXiv preprint arXiv:1707.06887* (2017).

[11] Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. "The cramer distance as a solution to biased wasserstein gradients." In: *arXiv preprint arXiv:1705.10743* (2017).

[12] Richard Bellman. "A Markovian decision process." In: *Journal of Mathematics and Mechanics* (1957), pp. 679–684.

[13] Carole Bernard and Steven Vanduffel. "Quantile of a mixture with application to model risk assessment." In: *Dependence Modeling* 3.1 (2015).

[14] Dimitri P Bertsekas and John N Tsitsiklis. "Neuro-dynamic programming: an overview." In: *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*. Vol. 1. IEEE. 1995, pp. 560–564.

[15] Peter J Bickel and David A Freedman. "Some asymptotic theory for the bootstrap." In: *The Annals of Statistics* (1981), pp. 1196–1217.

[16] Kang Boda and Jerzy A Filar. "Time consistent dynamic risk measures." In: *Mathematical Methods of Operations Research* 63.1 (2006), pp. 169–186.

[17] Vivek Borkar and Rahul Jain. "Risk-constrained Markov decision processes." In: *Decision and Control (CDC), 2010 49th IEEE Conference on*. IEEE. 2010, pp. 2664–2669.

[18] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[19] Yinlam Chow and Mohammad Ghavamzadeh. "Algorithms for CVaR optimization in MDPs." In: *Advances in neural information processing systems*. 2014, pp. 3509–3517.

[20] Yinlam Chow, Aviv Tamar, Shie Mannor, and Marco Pavone. "Risk-sensitive and robust decision-making: a CVaR optimization approach." In: *Advances in Neural Information Processing Systems*. 2015, pp. 1522–1530.

[21] Basel Committee et al. "Fundamental review of the trading book: A revised market risk framework." In: *Consultative Document, October* (2013).

[22] Stefano Paulo Coraluppi. "Optimal control of Markov decision processes for performance and robustness." In: (1998).

[23] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. "Distributional Reinforcement Learning with Quantile Regression." In: *arXiv preprint arXiv:1710.10044* (2017).

[24] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. *OpenAI Baselines*. https://github.com/openai/baselines. 2017.

[25] Javier Garcıa and Fernando Fernández. "A comprehensive survey on safe reinforcement learning." In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480.

[26] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[27] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. "Rainbow: Combining Improvements in Deep Reinforcement Learning." In: *arXiv preprint arXiv:1710.02298* (2017).

[28] Ronald A Howard and James E Matheson. "Risk-sensitive Markov decision processes." In: *Management science* 18.7 (1972), pp. 356–369.

[29] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. "Convergence of stochastic iterative dynamic programming algorithms." In: *Advances in neural information processing systems*. 1994, pp. 703–710.

[30] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. "What is the best multi-stage architecture for object recognition?" In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 2146–2153.

[31] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[32] Roger Koenker and Kevin F Hallock. "Quantile regression." In: *Journal of economic perspectives* 15.4 (2001), pp. 143–156.

[33] Vijay R Konda and John N Tsitsiklis. "Actor-critic algorithms." In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[34] Erwin Kreyszig. *Introductory functional analysis with applications*. Vol. 1. wiley New York, 1989.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[36] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[37] Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. "AI Safety Gridworlds." In: *arXiv preprint arXiv:1711.09883* (2017).

[38] Anirudha Majumdar and Marco Pavone. "How Should a Robot Assess Risk? Towards an Axiomatic Theory of Risk in Robotics." In: *arXiv preprint arXiv:1710.11040* (2017).

[39] Christopher W Miller and Insoon Yang. "Optimal control of conditional value-at-risk in continuous time." In: *SIAM Journal on Control and Optimization* 55.2 (2017), pp. 856–884.

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning." In: *Nature* 518.7540 (2015), p. 529.

[41] Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. "Nonparametric return distribution approximation for reinforcement learning." In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 799–806.

[42] Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. "Parametric return density estimation for reinforcement learning." In: *arXiv preprint arXiv:1203.3497* (2012).

[43] Georg Ch Pflug and Alois Pichler. "Time-consistent decisions and temporal decomposition of coherent risk functionals." In: *Mathematics of Operations Research* 41.2 (2016), pp. 682–699.

[44] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. "Parameter space noise for exploration." In: *arXiv preprint arXiv:1706.01905* (2017).

[45] LA Prashanth. "Policy gradients for CVaR-constrained MDPs." In: *International Conference on Algorithmic Learning Theory*. Springer. 2014, pp. 155–169.

[46] Herbert Robbins and Sutton Monro. "A stochastic approximation method." In: *The annals of mathematical statistics* (1951), pp. 400–407.

[47] R Tyrrell Rockafellar and Stanislav Uryasev. "Optimization of conditional value-at-risk." In: *Journal of risk* 2 (2000), pp. 21–42.

[48] R Tyrrell Rockafellar and Stanislav Uryasev. "Conditional value-at-risk for general loss distributions." In: *Journal of banking & finance* 26.7 (2002), pp. 1443–1471.

[49] Sebastian Ruder. "An overview of gradient descent optimization algorithms." In: *arXiv preprint arXiv:1609.04747* (2016).

[50] Yun Shen, Michael J Tobia, Tobias Sommer, and Klaus Obermayer. "Risk-sensitive reinforcement learning." In: *Neural computation* 26.7 (2014), pp. 1298–1328.

[51] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of go without human knowledge." In: *Nature* 550.7676 (2017), p. 354.

[52] Matthew J Sobel. "The variance of discounted Markov decision processes." In: *Journal of Applied Probability* 19.4 (1982), pp. 794–802.

[53] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

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

[55] Aviv Tamar, Yonatan Glassner, and Shie Mannor. "Optimizing the CVaR via Sampling." In: *AAAI*. 2015, pp. 2993–2999.

[56] Aviv Tamar, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. "Sequential decision making with coherent risk." In: *IEEE Transactions on Automatic Control* 62.7 (2017), pp. 3323–3338.

[57] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[58] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. "Dueling network architectures for deep reinforcement learning." In: *arXiv preprint arXiv:1511.06581* (2015).

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

[60] Evert Wipplinger. "Philippe Jorion: Value at Risk-The New Benchmark for Managing Financial Risk." In: *Financial Markets and Portfolio Management* 21.3 (2007), p. 397.

DECLARATION

Put your declaration here.

Prague, May, 2018

Silvestr Stanko

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Thank you very much for your feedback and contribution.

Final Version as of May, 2018 (`classicthesis` version 1.0).