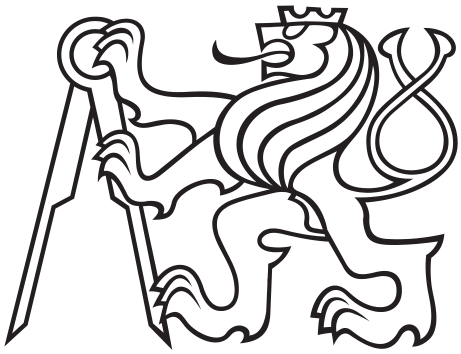
RISK-AVERSE DISTRIBUTIONAL REINFORCEMENT LEARNING

A CVAR OPTIMIZATION APPROACH

SILVESTR STANKO



Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University

May, 2018

[May 2, 2018 at 17:10 – classicthesis version 0.0]

ABSTRACT

Short summary of the contents in English. . . a great guide by Kent Beck how to write good abstracts can be found here:

https://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html

ABSTRAKT

Český abstrakt

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— **knuth:1974** [**knuth:1974**]

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

LISTINGS

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 [26] to machine translation [4]. Reinforcement learning has also been a part of this success, with excelent results regarding human-level control in computer games [31] or beating the best human players in the game of Go [41]. 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 [**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 if 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 [1, 28] directly tied to the discrepancies between the environment the agent trains on and is tested on. Chow et al. [17] 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.

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 it's 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 [18]. The metric also satisfies the recently proposed axioms of risk in robotics [29].

Another motivational point, aside from robustness, 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, or directly uses, the recent advances in distributional reinforcement learning [7, 19].

1.2 THESIS OUTLINE

1.3 CONTRIBUTIONS

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 basic theoretical foundations in Section 2.1, we remind the reader of the basic notions of reinforcement learning in Section 2.2 and of the recently explored and useful distributional reinforcement learning in Section **??**. We follow up with the basics of risk together with the crucial CVaR measure in Section 2.4.

The interested reader is welcome to explore the books and publications referenced throughout this chapter and in Section 2.6. An informed reader may choose to skip to Section 2.5 where we formalize the problems tackled in this thesis.

2.1 PROBABILITY ESSENTIALS

unclear: define random variable and expectation?

2.2 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 [43] 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.

2.2.1 *Markov Decision Processes*

Markov Decision Process (MDP, Bellman [9]) 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 math-

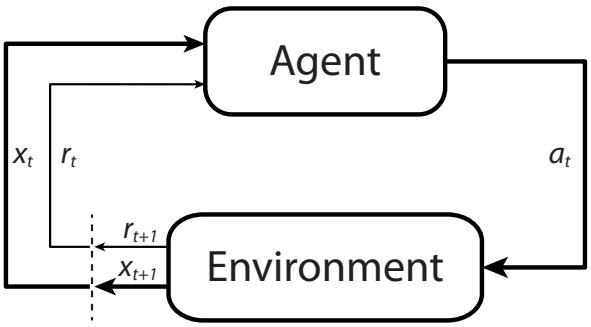


Figure 2.1: The Reinforcement learning cycle

ematically 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 Markov 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 1. *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*

$R(x, a) \in [R_{\min}, R_{\max}]$ *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 $t + 1$. **unclear:** weird? unify?

Definition 2. *A stationary (or Markovian) policy is a mapping from states to actions* $\pi : \mathcal{X} \to \mathcal{A}$*.*

todo: deterministic cost doesn't go together with distributional :(

2.2.2 *Return*

The ultimate goal of a 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, or 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:

$$Z^\pi(x) = \sum_{t=0}^{\infty} \gamma^t R(x_t, a_t) \qquad (2.1)$$
$$x_t \sim p(\cdot|x_{t-1}, a_{t-1}), a_t \sim \pi, x_0 = x$$

1 All the presented results are extendable to the case with random reward and with reward being a function of the transition $R(x, a, a')$. We avoid these extension to keep the results readable.

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.

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

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

2.2.3 *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:

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

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

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

Fundamental to reinforcement learning is the use of Bellman's equation [9] to describe the value and action-value functions by a recursive relationship:

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

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

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

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

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 expected reward function as one such vector. In this context, the *Bellman operator* $\mathcal{T}^\pi$ and *optimality operator* $\mathcal{T}$ are

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

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

These operators are useful as they describe the expected behaviour of popular learning algorithms such as SARSA and Q-Learning [43]. 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 [11].

2.2.4 *Contraction*

unclear: use something else than X for sets? An important concept used in convergence analysis of reinforcement learning algorithms is that of a contraction.

Definition 3. *A **fixed point** of a mapping $T : X \to X$ of a set X into itself is $x \in X$ which is mapped onto itself, that is $Tx = x$.*

*Let $X = (X, d)$ be a metric space (d is a metric on X). A mapping T is called a **contraction** on X if there exists a positive real number $\gamma < 1$ such that for all $x, y \in X$ $d(Tx, Ty) \leq \gamma d(x, y)$.*

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

See e.g. Kreyszig [25] for a complete treatment of all mentioned concepts.

The takeaway for reinforcement learning is, that if T is a contraction, by recursively applying $x_{k+1} = Tx_k$ 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.

2.3 DISTRIBUTIONAL REINFORCEMENT LEARNING

In contrast to standard reinforcement learning, where we model the expected return, in distributional reinforcement learning [**many**] 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 [33][32]. Bellemare, Dabney, and Munos [7] 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 [31]).

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 [43] 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 view the reward function as a random vector $R \in \mathcal{Z}$, and define the transition operator $P^\pi : \mathcal{Z} \to \mathcal{Z}$

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

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

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

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

Bellemare, Dabney, and Munos [7] have shown, that the distributional bellman operator $\mathcal{T}^\pi$ is not a contraction in the commonly used KL divergence [27], 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 [7] and Dabney et al. [19].

2.3.1 *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 [12]). The metric has recently gained in popularity and was used e.g. in unsupervised learning [2, 8]. 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).

unclear: maybe visuals, if it becomes important

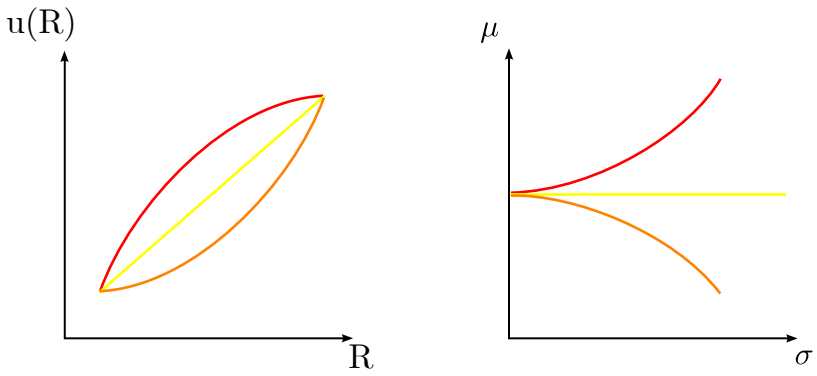


Figure 2.2: Standard in economic literature, these figures depict the differences between risk-averse (red), risk-neutral (yellow) and risk-seeking (orange) behaviors.
Left: A subjective utility $u(R)$, based on the reward, is concave for risk-averse behaviors. Right: Risk-averse utility contour lines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

2.4 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 this investment strategy is 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 [50], citepbasel2013fundamental. It has also been suggested in the neuroscience literature, that humans use risk-averse behaviors when making decisions [40].

As we stated in Section 1.1 and will formally state in Section 2.5, 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.

2.4.1 *Value-at-Risk*

Value-at-risk (VaR, see e.g. Wipplinger [50]) 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 rather than losses 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.

2.4.2 *Conditional Value-at-Risk*

Conditional Value-at-Risk (CVaR, see Rockafellar and Uryasev [37, 38]), 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. [3]), which makes CVaR computation much easier compared to VaR. It's properties were also recently identified as suitable for measuring risk in robotics [29]. 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 [18] and beyond [**something**].

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):

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

Rockafellar and Uryasev [37] also showed that CVaR is equivalent to the solution of

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

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

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

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

$$\mathrm{CVaR}_\alpha(Z) = \min_{\xi \in \mathcal{U}_{\mathrm{CVaR}}(\alpha, P(\cdot|x,a))} \mathbb{E}_\xi[Z] \tag{2.19}$$

$$\mathcal{U}_{\mathrm{CVaR}}(\alpha, P(\cdot|x,a)) = \left\{\xi : \xi(\omega) \in \left[0, \frac{1}{\alpha}\right], \int_\omega \xi(\omega)\mathbb{P}(\omega)\mathrm{d}\omega = 1\right\} \tag{2.20}$$

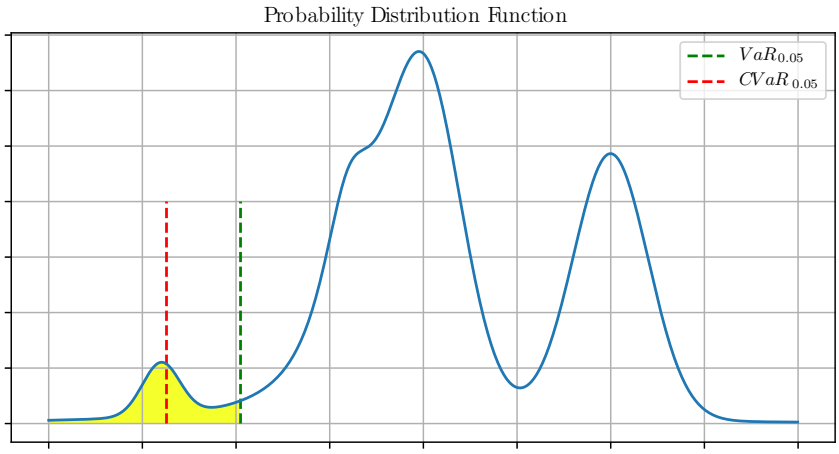


Figure 2.3: Value-at-Risk and Conditional Value-at-Risk of a general probability distribution with the integral $\alpha = 0.05$ marked in yellow. The main flaw of the VaR metric is clearly visible here, as we could shift the leftmost 'mode' of the distribution into minus infinity and the VaR would remain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

For a treatment of duality, see e.g. Boyd and Vandenberghe [15].

2.5 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 4 (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 the following

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

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.

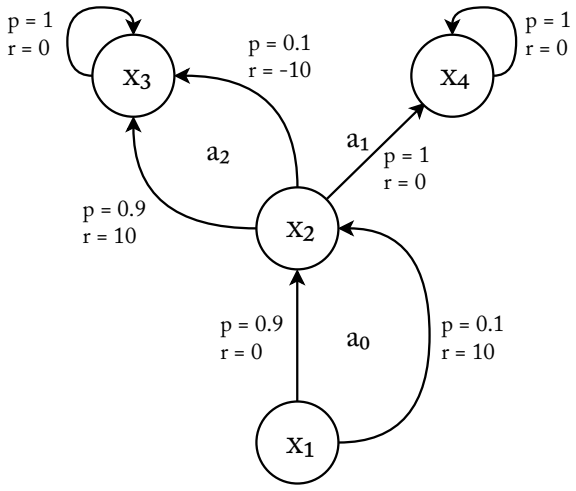


Figure 2.4: MDP showing time-inconsistency of the CVaR objective.

2.5.1 *Time-consistency*

There exist several definitions of time-consistency [13, 34]. Informally, if the criterion is time-consistent, we can limit ourselves to the space of stationary policies, as the 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 time-inconsistent. On 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 $\alpha = 0.19$:

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

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.

unclear: this is for $\gamma = 1$, different gamma would introduce ugly numbers **todo:** mention the α inconsistency part

2.5.2 *Robustness*

An important motivational point for the CVaR objective (2.21) is it's relationship with robustness. Chow et al. [17] 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 [17])

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

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 $\hat{P}_t$.

2.6 LITERATURE SURVEY

Risk-sensitive MDPs have been studied thoroughly in the past, with different risk-related objectives. Due to it's good computational properties, earlier efforts focused on exponential utility [21], the max-min criterion [**minmax**] or e.g. maximizing the mean with constrained variance [42]. A comprehensive overview of the different objectives can be found in Garcıa and Fernández [20], 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 [14, 35].

Recently, for the reasons explained above, several authors have investigated the exact objective (2.21). A considerable effort has gone towards policy-gradient [44] and Actor-Critic [24] algorithms with the CVaR objective. Chow and Ghavamzadeh [16] and Tamar, Glassner, and Mannor [45] 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 [47] and a gradient-based Actor-Critic algorithm [47]. 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 [30].

Morimura et al. [32, 33] were among the first to utilize distributional reinforcement learning with both parametric and npnparametric models and used it to optimize CVaR, however they only used the naive approach discussed in Section 2.5.1.

Bäuerle and Ott [6] 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. [17] also uses a continuous augmented state-space but unlike Bäuerle and Ott [6], 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 a linear program computation, which we address in the next chapter.

3

VALUE ITERATION WITH CVAR

Value iteration is a standard algorithm for maximizing expected discounted reward used in reinforcement learning. In this chapter we extend the results of Chow et al. [17], who have recently proposed an approximate value iteration algorithm for CVaR MDPs.

The original algorithm requires the 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 algorithm, we present the original algorithm in Section 3.1. The improved algorithm is presented in Section 3.2, followed by section Section 3.3, where we test the algorithm on selected environments.

3.0.1 *Value Iteration*

Value iteration [43] is a well-known algorithm for computing the optimal (action-)value function and hereby finding the optimal policy. Let us remind ourselves of the Bellman optimality operator $\mathcal{T}$ (2.9):

$$\mathcal{T}Q(x,a) := \mathbb{E}\left[R(x,a)\right] + \gamma\, \mathbb{E}_P\left[\max_{a' \in \mathcal{A}} Q(x',a')\right]$$

or rewritten for the value function V

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

As stated before, $\mathcal{T}$ is a contraction (Section 2.2.4). 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**
    $V(x) = \max_a \{r(x,a) + \gamma \sum_{s'} p(s'|x,a)V(x')\}$
    $\Delta = \max\{\Delta, |v - V(x)|\}$
  end for
until $\Delta < \epsilon$
Output a deterministic policy $\pi \approx \pi^*$:
  $\pi(x) = \arg\max_a \{r(x,a) + \gamma \sum_{s'} p(s'|x,a)V(x')\}$

3.1 CVAR VALUE ITERATION

Chow et al. [17] 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. [17] proved it's 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.

3.1.1 *Bellman Equation for CVaR*

The results of Chow et al. [17] heavily rely on the CVaR decomposition theorem (Lemma 22, [34]):

$$\mathrm{CVaR}_\alpha\left(Z^\pi(x)\right) = \min_{\xi \in \mathcal{U}_{\mathrm{CVaR}}(\alpha, P(\cdot|x,a))} \sum_{x'} p(x'|x,\pi(x))\xi(x')\mathrm{CVaR}_{\xi(x')\alpha}\left(Z^\pi(x')\right) \quad (3.2)$$

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 (or worst-case) weighted combination of $\mathrm{CVaR}_\alpha\left(Z^\pi(x')\right)$ under a probability distribution perturbed by $\xi(x')$.

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. [17] 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.

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

1 We use C instead of V in our notation.

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:

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

Chow et al. [17] further showed (Lemma 3) that the operator $\mathbf{T}$ is a contraction and also preserves the convexity of $y\text{CVaR}_t$. 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 $C^*(x,y) = \max_{\pi \in \Pi} \text{CVaR}_y(Z^\pi(x,y))$ ([17], 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 [17] is then to represent the convex $y\text{CVaR}_y$ as a piecewise linear function.

3.1.2 *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 interpolate the $yC(x,y)$ function 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 ([17], Theorem 7). **todo:** bounded -> linear in θ

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

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.

unclear: maybe formulate the LP exactly?

3.1.3 *Optimal policy*

An important product of any value iteration algorithm is the optimal policy. The value-function C^* can be used to extract the optimal policy π^* of the original problem (2.21), using the following theorem.

Algorithm 2 CVaR Value Iteration with Linear Interpolation (Algorithm 1 in [17])

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 $y_i < y_{i+1}$, $y_1 = 0$ 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 $y_i \in \mathbf{Y}(x)$.

Theorem 2 (Optimal Policies, Theorem 5 in [17])**.** ***todo:*** *k->t Let $\pi_H^* = \{\mu_0, \mu_1, \ldots\} \in \Pi_H$ be a history-dependent policy recursively defined as:*

$$\mu_k(h_k) = u^*(x_k, y_k), \ \forall k \geq 0, \tag{3.7}$$

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

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

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 this holds for both the original operator $\mathbf{T}$ and the linearly interpolated $\mathbf{T}_{\mathcal{I}}$. Chow et al. [17] further showed that the error is bounded

3.2 EFFICIENT COMPUTATION USING QUANTILE REPRESENTATION

We present our original contributions in this section. We first describe 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

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

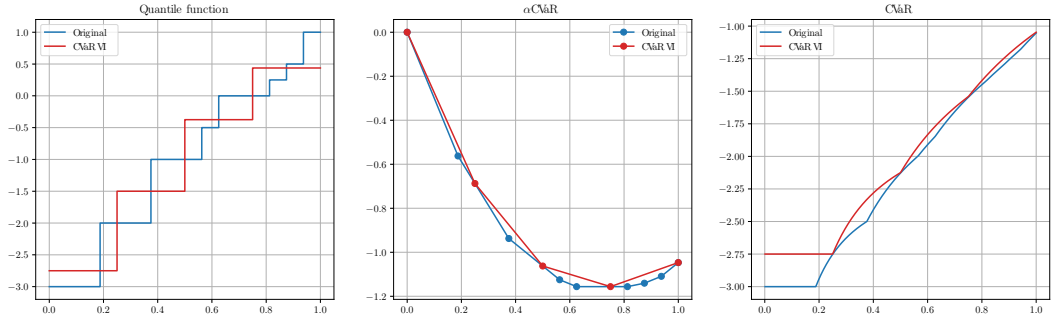


Figure 3.1: Comparison of a discrete distribution and it's approximation according to the CVaR linear interpolation operator.

and the derivative of the yCVaR$_y$ function is the quantile function

$$\frac{\partial}{\partial y} y\mathit{CVaR}_y(Z) = \mathit{VaR}_y(Z) \tag{3.10}$$

Proof. The fact that discrete distributions have a piecewise linear yCVaR$_y$ function has already been shown by Rockafellar and Uryasev [37].
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 yCVaR$_y$ 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.

3.2.1 *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 αCVaR$_\alpha$ function to compute CVaR at each atom. The general steps of the computation are as follows

1. transform yCVaR$_y$ of each possible state transition to a discrete probability distribution using (3.10)
2. combine these to to a distribution representing the full state-action distribution
3. compute yCVaR$_y$ for all atoms using (3.9)

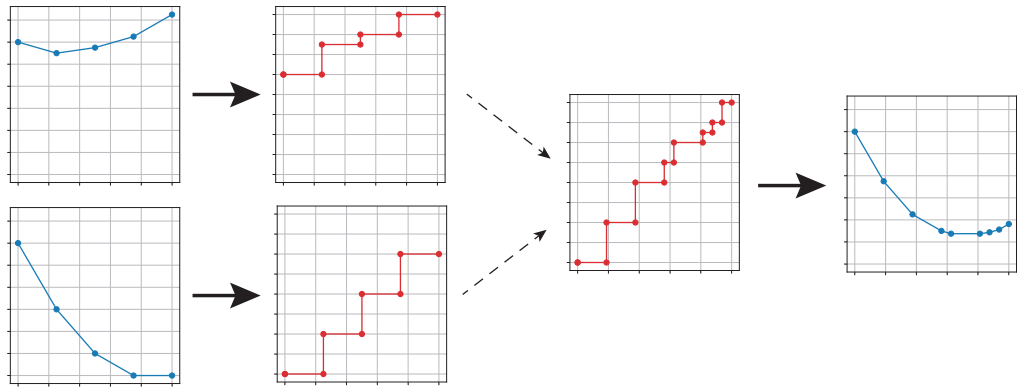


Figure 3.2: Visualization of the CVaR computation for a single state and action with two transition states. Thick arrows represent the conversion between $y\text{CVaR}_y$ and the quantile function.

See Figure 3.2 for a visualization of the procedure.
Note that this procedure is linear for discrete distributions. For completeness, we show the explicit computation in **appendix**. **todo:** appendix

The proposed method draws remarkable similarity to distributional RL (Section 2.3). 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 contrary to CVaR VI, where 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 VI, we construct the 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 the problem (3.2) in the next section. Note that we skip the reward and gamma scaling for readability's sake. The extension to the bellman operator is trivial.

3.2.2 *ξ-computation*

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

Theorem 3. *Let x'_1, x'_2 be two states reachable from state x 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*

$$\xi(x'_i) = \frac{F_{x'_i}(F_x^{-1}(\alpha))}{\alpha} \tag{3.11}$$

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

$$\begin{aligned}\alpha\text{CVaR}_\alpha(Z(x)) = \min_{\xi}\ & p_1\xi_1\alpha\text{CVaR}_{\xi_1\alpha}(Z(x_1')) + p_2\xi_2\alpha\text{CVaR}_{\xi_2\alpha}(Z(x_2')) \\ \text{s.t.}\quad & p_1\xi_1 + p_2\xi_2 = 1 \\ & 0 \le \xi_1 \le \frac{1}{\alpha} \\ & 0 \le \xi_2 \le \frac{1}{\alpha}\end{aligned}$$

therefore

$$\begin{aligned}\alpha\text{CVaR}_\alpha(Z(x)) &= \min_{\xi}\ p_1\xi_1\alpha\text{CVaR}_{\xi_1\alpha}(Z(x_1')) + (1-p_1)\frac{1-p_1\xi_1}{1-p_1}\alpha\text{CVaR}_{\frac{1-p_1\xi_1}{1-p_1}\alpha}(Z(x_2')) \\ &= \min_{\xi}\ p_1\int_0^{\xi_1\alpha}\text{VaR}_\beta(Z(x_1'))\,\mathrm{d}\beta + (1-p_1)\int_0^{\frac{1-p_1\xi_1}{1-p_1}\alpha}\text{VaR}_\beta(Z(x_2')\,\mathrm{d}\beta\end{aligned}$$

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

$$\begin{aligned}\text{VaR}_{\xi_1\alpha}(Z(x_1')) = \text{VaR}_{\xi_2\alpha}(Z(x_2')) &= \text{VaR}_\alpha(Z(x)) \\ F^{-1}_{Z(x_1')}(\xi_1\alpha) = F^{-1}_{Z(x_2')}(\xi_2\alpha) &= F^{-1}_{Z(x)}(\alpha)\end{aligned}$$

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

$$\begin{aligned}F^{-1}_{Z(x_1')}(\xi_1\alpha) &= F^{-1}_{Z(x)}(\alpha) && / F_{Z(x_1')} \\ F_{Z(x_1')}\left(F^{-1}_{Z(x_1')}(\xi_1\alpha)\right) &= F_{Z(x_1')}\left(F^{-1}_{Z(x)}(\alpha)\right) \\ \xi_1\alpha &= F_{Z(x_1')}\left(F^{-1}_{Z(x)}(\alpha)\right)\end{aligned}$$

And similarly for ξ_2.

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

□

The theorem is ***easily*** extended 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 [10] for details on the two-dimensional case for general distributions.

3.3 EXPERIMENTS

We test the proposed algorithm on the same task as Chow et al. [17]. The task of the agent is to navigate on a rectangular grid to a given destination, moving in it's 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 [17]. The learned policies on a sample grid are shown in Figure 3.3.

Chow et al. [17] report computation time on the order of two hours, our naive Python implementation converged within 20 minutes and there is ample room for improvement.

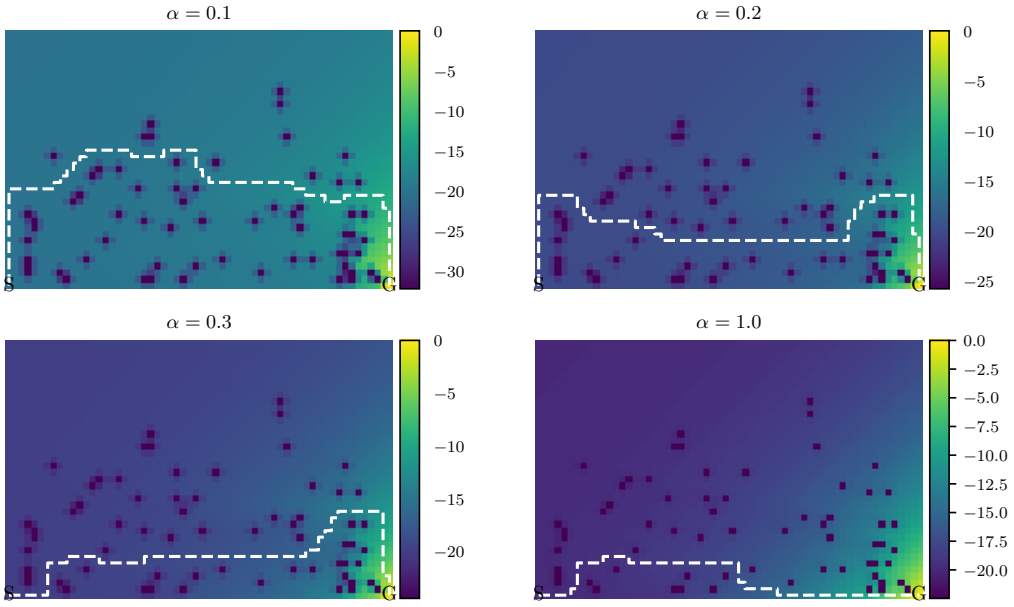


Figure 3.3: Grid-world simulations. The optimal deterministic paths are shown together with CVaR estimates for given αs.

3.3.1 *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 unoptimal 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 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. **unclear:** seems that the error is unbounded in case of QR-DQN, go into more detail? provide examples?

These problems can always resolved by increasing the number of atoms around these critical points, as both methods are ***valid?*** in the limit. Whether one can mitigate these problems without the need for adding more atoms remains an open question.

3.4 SUMMARY

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 4.0.1 and introduce CVaR estimation in Section 4.0.2. These concepts are combined together with CVaR value iteration and in Section 4.1 we propose the first CVaR Q-learning algorithm. We treat the optimal policy separately in Section 4.2.

The algorithm is then experimentally verified on suitable environments in Section 4.3.

4.0.1 *Q-learning*

Q-learning (Watkins and Dayan [49]) 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.

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

Here Q is an estimate of the optimal action-value function (2.7) and β is the learning rate. 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 [22] 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. See Algorithm 3 for a full practical procedure.

4.0.2 *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.

Algorithm 3 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'
    $Q(x, a) = (1 - \beta)Q(x, a) + \beta \left[r + \gamma \max_{a'} Q(x', a')\right]$
    $x = x'$
  end while
end for

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 it's 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 [36].

The RM theorem has also been applied directly to CVaR estimation by Bardou, Frikha, and Pages [5], 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 a 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.

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

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

An observant reader may recognize a standard equation for quantile estimation in equation (4.3) (see e.g. Koenker and Hallock [23] for more information on quantile estimation/regression) and 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) [5].

4.1 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.

4.1.1 *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 one-step VaR_y estimate, or the estimate of the $y-$quantile of a distribution recovered from CVaR_y by Lemma 1.

A key to any temporal difference algorithm is it's update rule. The CVaR TD update rule extends the improved VI procedure and we present the full rule in Algorithm 4.

Let us now go through the algorithm and compare the two versions. We first construct a new CVaR (line 3), representing CVaR of $Z(x')$, by greedily selecting actions that yield the highest CVaR for each atom. This step is somewhat skipped in the VI procedure since we are not working with action-value functions. These values are then transformed to their underlying distributions (line 5) and used to generate samples from this distribution. Since we know the distribution estimates exactly, we do not have to actually sample - instead we use the quantile values proportionally to their probabilities and apply the respective VaR and CVaR update rules (lines 7, 8).

Note that if the atoms aren't uniform, we perform basic importance sampling in the expectation terms, weighting each 'sample' according to it's probability.

Algorithm 4 CVaR TD update

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

4.1.2 *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 procedure. The update rules (**??**) have been only shown to converge if the underlying distributions are continuous [**...**], which is not the case in our setting.

It is also unclear if the joint procedure converges. ***check the random processes convergence***

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

4.2 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 VI, since we would have to have acces 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.

4.2.1 *VaR-based Policy Improvement*

We first introduce the VaR-based policy improvement in the context of distributional RL and extend it to CVaR Q-learning in the next section.

Let us now assume that we have succesfully converged with distributional value iteration and therefore have available the return distributions of some stacionary policy for each state and action. Our next goal is to find a policy improvement algorithm that will monotonically increase the αCVaR_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

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

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

$$\max_\pi CVaR_\alpha(Z^\pi) = \max_\pi \frac{1}{\alpha} \mathbb{E}\left[(Z^\pi - s^*)^-\right] + s^* \tag{4.5}$$

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 $s^* = VaR_\alpha(Z(x_0))$ and focus on maximization of the simpler criterion.

This can be seen as coordinate ascent; in the first phase (when we compute CVaR) we maximize w.r.t. s while keeping π fixed; in the second phase we fix s and maximize w.r.t. π.

The extension to MDPs consists of recursively updating the s we maximize by setting $s = \frac{s - r_t}{\gamma}$. See Algorithm 5 for the full procedure which we justify in the following theorem.

Algorithm 5 VaR-based policy improvement

$a = \arg\max_a CVaR_\alpha(Z(x_0, a))$
$s = VaR_\alpha(Z(x_0, a))$
$x_t, r_t = \text{envTransition}(x_0, a)$
while x_t is not terminal **do**
  $s = \frac{s - r_t}{\gamma}$
  $a = \arg\max_a \mathbb{E}\left[(Z(x_t, a) - s)^-\right]$
  $x_t, r_t = \text{envTransition}(x_t, a)$
end while

Theorem 4. *Let π be a stationary policy, $\alpha \in (0, 1]$. By following policy π' from algorithm 5, we improve $CVaR_\alpha(Z)$ in expectation:*

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

Proof. Let s^* be a solution to (2.17). Then by optimizing $\max_\pi \frac{1}{\alpha}\mathbb{E}\left[(Z^\pi - s^*)^-\right]$, we monotonely improve the optimization criterion (2.21).

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

We can rewrite our subgoal as

$$\begin{aligned} \mathbb{E}\left[(Z_t - s)^-\right] &= \mathbb{E}\left[(Z_t - s)\mathbb{1}(Z_t \leq s)\right] = \mathbb{E}\left[(R_t + \gamma Z_{t+1} - s)\mathbb{1}(Z_{t+1} \leq \frac{s - R_t}{\gamma})\right] \\ &= \sum_{x_{t+1}, r_t} P(x_{t+1}, r_t \mid x_t, a)\mathbb{E}\left[(r_t + \gamma Z(x_{t+1}) - s)\mathbb{1}(Z(x_{t+1}) \leq \frac{s - r_t}{\gamma})\right] \\ &= \sum_{x_{t+1}, r_t} P(x_{t+1}, r_t \mid x_t, a)\mathbb{E}\left[\gamma\left(Z(x_{t+1}) - \frac{s - r_t}{\gamma}\right)\mathbb{1}(Z(x_{t+1}) \leq \frac{s - r_t}{\gamma})\right] \\ &= \gamma \sum_{x_{t+1}, r_t} P(x_{t+1}, r_t \mid x_t, a)\mathbb{E}\left[\left(Z(x_{t+1}) - \frac{s - r_t}{\gamma}\right)\mathbb{1}(Z(x_{t+1}) \leq \frac{s - r_t}{\gamma})\right] \\ &= \gamma \sum_{x_{t+1}, r_t} P(x_{t+1}, r_t \mid x_t, a)\mathbb{E}\left[\left(Z(x_{t+1}) - \frac{s - r_t}{\gamma}\right)^-\right] \end{aligned} \tag{4.6}$$

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 $\hat{r}_t$ and state $\hat{x}_{t+1}$, we are still trying to find a policy π^* that maximizes

$$\begin{aligned} \pi^* &= \arg\max_\pi \mathbb{E}\left[(Z_t - s)^- \middle| \hat{x}_{t+1}, \hat{r}\right] \\ &= \arg\max_\pi \mathbb{E}\left[\left(Z(\hat{x}_{t+1}) - \frac{s - \hat{r}_t}{\gamma}\right)^-\right] \end{aligned} \tag{4.7}$$

Where, by the linearity of the expected value, we ignored the unsampled states (since these are not a function of $\hat{x}_{t+1}$) 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 **??**. By induction we maximize the criterion (4.5) 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.

4.2.2 *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. Unfortunately we cannot straightforwardly maximize $\mathbb{E}\left[(Z_t - s^*)^-\right]$ since the Value-at-Risk estimates V we have do not reflect the actual VaR of the return. Instead they represent the one-step VaR of a distribution extracted from the optimal CVaR.

To side-step this issue, we bring the VaR-based improvement closer to CVaR VI and use it only in a one-step context. In each step, we compute the next y as $F_{Z(x)}(\frac{s-r}{\gamma})$, which we extract from our VaR estimate V. Since we don't have access to the exact VaR for each $y \in [0, 1]$, we use linear interpolaton as a heuristic. See Algorithm 6 for the full procedure.

Algorithm 6 CVaR Q-learning policy

input: α, converged $V(x, a, y), C(x, a, y)$
$x = x_0$
$y = \alpha$
$a = \arg\max_a C(x, a, y)$
$s = V(x, a, y)$
while x is not terminal **do**
  Take action a, observe r, x'
  $y = F_{Z(x')}(\frac{s-r}{\gamma})$
  $a = \arg\max_a C(x, a, y)$
  $s = V(x, a, y)$
  $x = x'$
end while

todo: end section, full cvar q?

4.3 EXPERIMENTS

We use the same environment from Section **??**. Since the positive reward is very sparse, we chose to run CVaR Q-learning on a smaller environment of size 10×15.

- atom spacing

Algorithm 7 Full CVaR Q-learning

Initialize $V(x,a,y), C(x,a,y)$ for all $x \in \mathcal{X}, a \in \mathcal{A}, y \in \mathcal{Y}$ arbitrarily, and $C(x_{\text{terminal}}, \cdot, \cdot) = 0$
for each episode **do**
  $x = x_0$
  $a_0 = \arg\max_a \text{CVaR}_\alpha(x,a)$
  $s = \text{VaR}_\alpha(x, a_0)$
  while x is not terminal **do**
    Choose a using a policy derived from C, s (e.g. ε-greedy)
    Take action a, observe r, x'
    $UPDATE(x, a, x', r)$
    $s = \frac{s - r}{\gamma}$
    $x = x'$
  end while
end for

- all parameters in appendix
- extraction via different policies (plots?)
- running with different policies

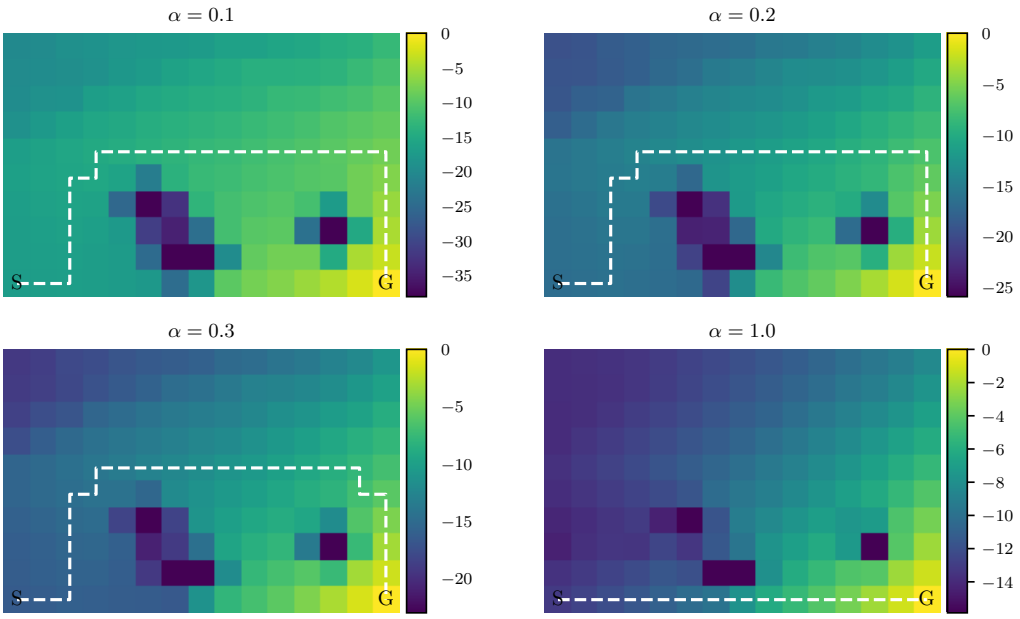


Figure 4.1: Grid-world Q-learning simulations. The optimal deterministic paths are shown together with CVaR estimates for given αs.

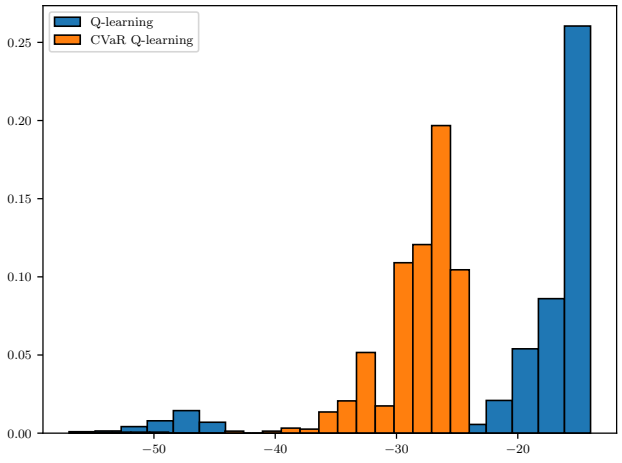


Figure 4.2: Sample histograms for policies generated by Q-learning and CVaR Q-learning with $\alpha = 0.1$.

5

DEEP CVAR Q-LEARNING

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. We follow the work of Mnih et al. [31] proposed the Deep Q-learning (DQN) algorithm and succesfully trained on multiple different high-dimensional environments, resulting in the first artificial agent that is capable of learning to excel at a diverse array of challenging tasks.

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

5.1 DEEP Q-LEARNING

The ultimate goal of artificial inteligence are agents that perform well across wide range of tasks. In the past, research was often focused on narrow AI, which was able to perform well on one particular task, but was useless on others. That changed with the advent of deep learning [**neco**], that allowed us to train function approximators with the ability to generalize. Deep learning methods have become popular across all of machine learning, especially for supervised learning and vision.

unclear: neural networks? but its common knowledge

Deep learning has also been succesfully applied to reinforcement learning. Q-learning have been used with function approximators in the past[**neco**], 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 [**...**] and this has been a problem in practice as well. Mnih et al. [31] 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 [**atari**], 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.

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 $T_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} Q'(x'_j, a') & \text{otherwise} \end{cases}$
    Perform a gradient step on $(T_j - Q(x_j, a_j))$ w.r.t. θ
    Every N_{target} steps set $\theta' = \theta$
  end while
end for

5.2 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. [19], which shares certain similarities with CVaR Q-learning.

In QR-DQN, which follows up on the theoretical results of Bellemare, Dabney, and Munos [7], the goal is to learn a distribution that minimizes the wasserstein distance from the actual return distribution. The distributions are represented as discrete uniform probability atoms and the ***

...

5.3 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:

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

First significant change against 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 functions.

In DQN, we minimize the mean squared error (MSE) of $\mathbb{E}\left[\left(\mathcal{T}Q(x, a) - Q(x, a) \right)^2 \right]$ as the true $Q(x, a)$ is the minimal argument of MSE.

In QR-DQN we instead optimize the quantile loss function $\mathbb{E}_j \left[(\mathcal{T}V_j - V_i)(y_j - \mathbb{1}_{(V_i \geq \mathcal{T}V_j)}) \right]$.
In CVaR-DQN we have to fi***

5.3.1 *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 distribution however is constructed differently - in CVaR-DQN we extract the distribution from the $y\text{CVaR}_y$ function of the next state.

$$\mathcal{L}_{\text{VaR}} = \sum_{i=1}^{N} \mathop{\mathbb{E}}_{j} \left[(r + \gamma d_j - V_i(x, a))(y_j - \mathbb{1}_{(V_i(x,a) \geq r + \gamma d_j)}) \right] \tag{5.1}$$

where d_j are the extracted atoms.

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

$$\mathcal{L}_{\text{CVaR}} = \sum_{i=1}^{N} \mathop{\mathbb{E}}_{j} \left[\left(V_i(x, a) + \frac{1}{y_i} (r + \gamma d_j - V_i(x, a))^{-} - C_i(x, a) \right)^2 \right] \tag{5.2}$$

A perhaps simpler loss function can be constructed by considering the formulation of CVaR_α as the expected value before VaR_α, leading to the following loss function

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

This loss can run into trouble if we had multiple atoms with the same values, however we found **todo:** try this that both loss functions performed identically in our experiments.

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

Algorithm 9 Deep CVaR Loss function

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

5.3.2 *CVaR-DQN with experience replay*

5.4 EXPERIMENTS

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

6

CONCLUSION

Bäuerle and Ott [6] Bellemare, Dabney, and Munos [7] Chow et al. [17] Dabney et al. [19] Garcıa and Fernández [20] Majumdar and Pavone [29] Morimura et al. [32] Morimura et al. [33] Pflug and Pichler [34] Rockafellar and Uryasev [37] Rockafellar and Uryasev [38] Majumdar and Pavone [29] Leike et al. [28] Amodei et al. [1] Shapiro [39] Artzner et al. [3] Tamar et al. [46] Sutton and Barto [43] Watkins and Dayan [49] Bellman [9] Tsitsiklis [48] Boyd and Vandenberghe [15] Kreyszig [25] Koenker and Hallock [23] Committee [18] Mnih et al. [31] Silver et al. [41] Bahdanau, Cho, and Bengio [4] Krizhevsky, Sutskever, and Hinton [26]

A

VALUE ITERATION

LP
full VI

Algorithm 11 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**
    $V(x) = \max_a \{r(x,a) + \gamma \sum_{s'} p(s'|x,a)V(x')\}$
    $\Delta = \max\{\Delta, |v - V(x)|\}$
  end for
until $\Delta < \epsilon$
Output a deterministic policy $\pi \approx \pi^*$:
  $\pi(x) = \arg\max_a \{r(x,a) + \gamma \sum_{s'} p(s'|x,a)V(x')\}$

BIBLIOGRAPHY

[1] 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).

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

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

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

[5] 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.

[6] 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.

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

[8] 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).

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

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

[11] 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.

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

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

[14] 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.

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

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

[17] 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.

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

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

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

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

[22] 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.

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

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

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

[26] 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.

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

[28] 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).

[29] 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).

[30] 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.

[31] 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.

[32] 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.

[33] 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).

[34] 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.

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

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

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

[38] 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.

[39] Alexander Shapiro. "On Kusuoka representation of law invariant risk measures." In: *Mathematics of Operations Research* 38.1 (2013), pp. 142–152.

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

[41] 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.

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

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

[44] 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.

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

[46] Aviv Tamar, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. "Policy gradient for coherent risk measures." In: *Advances in Neural Information Processing Systems*. 2015, pp. 1468–1476.

[47] 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.

[48] John N Tsitsiklis. "Asynchronous stochastic approximation and Q-learning." In: *Machine learning* 16.3 (1994), pp. 185–202.

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

[50] 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.