# Risk-averse Distributional Reinforcement Learning: A CVaR Optimization Approach

Silvestr Stanko and Karel Macek
*DHL ITS Digital Lab, Czech Republic*

Keywords: Reinforcement Learning, Distributional Reinforcement Learning, Risk, AI Safety, Conditional Value-at-Risk, CVaR, Value Iteration, Q-learning, Deep Learning, Deep Q-learning.

Abstract: Conditional Value-at-Risk (CVaR) is a well-known measure of risk that has been directly equated to robustness, an important component of Artificial Intelligence (AI) safety. In this paper we focus on optimizing CVaR in the context of Reinforcement Learning (RL), as opposed to the usual risk-neutral expectation. As a first original contribution, we improve the CVaR Value Iteration algorithm (Chow et al., 2015) in a way that reduces computational complexity of the original algorithm from polynomial to linear time. Secondly, we propose a sampling version of CVaR Value Iteration we call CVaR Q-learning. We also derive a distributional policy improvement algorithm, 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, including the Deep CVaR Q-learning agent which learns how to avoid risk from raw pixels.



## 1 INTRODUCTION

Lately, there has been a surge of successes in machine learning research and applications, ranging from visual object detection (Krizhevsky et al., 2012) to machine translation (Bahdanau et al., 2014). Reinforcement learning has also been a part of this success, with excellent results regarding human-level control in computer games (Mnih et al., 2015) or beating the best human players in the game of Go (Silver et al., 2017). 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 (Hamid and Braun, 2019). This is partially caused by concerns of safety, as mistakes can be costly in the real world.

Robustness, or distributional shift, is one of the identified issues of AI safety (Leike et al., 2017) directly tied to the discrepancies between the environment the agent trains on and is tested on. (Chow et al., 2015) 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.

While the term risk is a general one, we will focus on one particular risk measure 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 (Committee et al., 2013) and it also satisfies the recently proposed axioms of risk in robotics (Majumdar and Pavone, 2017).

The aim of this paper 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 (Bellemare et al., 2017) (Dabney et al., 2017).

Risk-sensitive decision making in Markov Decicion Processes (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 (Howard and Matheson, 1972), the max-min criterion (Coraluppi, 1998) or e.g. maximizing the mean with constrained variance (Sobel, 1982). Some attempts were done with non-parametric VaR optimization (Macek, 2010). A comprehensive overview of the different objectives can be found in (Garcıa and Fernández, 2015), together



Stanko, S. and Macek, K.
Risk-averse Distributional Reinforcement Learning: A CVaR Optimization Approach.
DOI: 10.5220/0008175604120423
In *Proceedings of the 11th International Joint Conference on Computational Intelligence (IJCCI 2019)*, pages 412-423
ISBN: 978-989-758-384-1

with a unified look on the different methods used in safe reinforcement learning. From the more recent investigations, we can mention the approach considering safety as primary objective whereas the reward is secondary in the lexicographical sense (Lesser and Abate, 2017). Among CVaR-related objectives, some publications focus on optimizing the expected value with a CVaR constraint (Prashanth, 2014).

Recently, for the reasons explained above, several authors have investigated minimization of CVaR in Markov Decision Processes. A considerable effort has gone towards policy-gradient and Actor-Critic algorithms with the CVaR objective. (Tamar et al., 2015) 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., 2017) present a policy gradient algorithm and a gradient-based Actor-Critic algorithm.

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 (Miller and Yang, 2017).

(Morimura et al., 2012) were among the first to utilize distributional reinforcement learning with both parametric and nonparametric models and used it to optimize CVaR. (Dabney et al., 2018) also formulated a sampling-based approach for distributional RL and CVaR. However, all mentioned authors used only a naive approach that does not take into account the time-inconsistency of the CVaR objective.

(Bäuerle and Ott, 2011) used a state space extension and showed that this new extended state space contains globally optimal policies. Unfortunately, the state-space is continuous which brings more complexity.

The approach of (Chow et al., 2015) also uses a continuous augmented state-space but unlike (Bäuerle and Ott, 2011), 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 section.

The paper is organized as follows: In Section 2, we introduce the notation and define the addressed problem: the maximization of CVaR of the MDP return. Subsequent sections provide the original theoretical contributions: Section 3 improves a faster version of the state-of-art CVaR Value Iteration; Section 4 applies similar principles to situations where an exact model is not known and introduces CVaR Q-learning and describes CVAR policy improvement that can be used for the efficient extraction of policies; Section 5 extends CVaR Q-learning to its approximate variant using deep learning. Section 6 describes the conducted experiments that support the outlined original contributions. Finally, Section 7 concludes the paper and outlines the direction for further research.

All proofs and further materials can be found online[1].

# 2 PRELIMINARIES

## 2.1 Basic Notation

$\mathbb{P}(\cdot)$ denotes the probability of an event. We use $p(\cdot)$ and $p(\cdot|\cdot)$ for the probability mass function and conditional probability mass function respectively. The cumulative distribution function is defined as $F(z) = \mathbb{P}(Z \leq z)$. For the random variables we work with the expected value $\mathbb{E}[Z]$, Value at Risk

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

with confidence level $\alpha \in (0,1)$ and Conditional Value at Risk as

$$\mathrm{CVaR}_\alpha(Z) = \frac{1}{\alpha}\int_0^\alpha F_Z^{-1}(\beta)\mathrm{d}\beta = \frac{1}{\alpha}\int_0^\alpha \mathrm{VaR}_\beta(Z)\mathrm{d}\beta \quad (2)$$

*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 are based on the same underlying principles.

*CVaR as Optimization*: (Rockafellar and Uryasev, 2000) proved the following equality

$$\mathrm{CVaR}_\alpha(Z) = \max_s \left\{ \frac{1}{\alpha}\mathbb{E}\left[(Z-s)^-\right] + s \right\} \quad (3)$$

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

$$\mathrm{CVaR}_\alpha(Z) = \frac{1}{\alpha}\mathbb{E}\left[(Z - VaR_\alpha(Z))^-\right] + VaR_\alpha(Z). \quad (4)$$

[1] https://bit.ly/2EkXS0F

*CVaR Dual Formulation*: CVaR can be expressed also as:

$$\mathrm{CVaR}_\alpha(Z) = \min_{\xi \in \mathcal{U}_{\mathrm{CVaR}}(\alpha, p(\cdot))} \mathbb{E}_\xi[Z] \tag{5}$$

where

$$\mathcal{U}_{\mathrm{CVaR}}(\alpha, p(\cdot)) = \left\{ \xi : \xi(z) \in \left\{0, \tfrac{1}{\alpha}\right\}, \int \xi(z) p(z) \mathrm{d}z = 1 \right\} \tag{6}$$

We provide basic intuition behind the dual variables as these will become important later: In case of a discrete probability distribution, 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 $\mathrm{CVaR}_\alpha(Z)$. Values above $\mathrm{VaR}_\alpha(Z)$ are not taken into account so their $\xi$ is 0. If there exists an atom (i.e. a single $z$ with non-zero probability) at $\mathrm{VaR}_\alpha(Z)$, the variables are linearly interpolated to fit the constraints.

## 2.2 Markov Decision Process

Markov Decision Process (MDP, (Bellman, 1957)) 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 reward generated by being in state $x$ and selecting action $a$, $p(x'|x,a)$ is the probability of transition to new state $x'$ given state $x$ and action $a$. $\gamma \in [0,1)$ is a discount factor. A stationary policy $/pi$ is a mapping from states to probabilities of selecting each possible action $\pi : \mathcal{X} \times \mathcal{A} \to [0,1]$. For indexing the time, we use $t = 0, 1, \ldots, \infty$.

The *return* is defined as discounted sum of rewards over the infinite horizon, given policy $\pi$ and initial state $x_0$:

$$Z^\pi(x_0) = \sum_{t=0}^{\infty} \gamma^t r(x_t, a_t)$$

Note that the return is a random variable.

## 2.3 Problem Formulation

The problem tackled in this article 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:

**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 distribution of 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 $\alpha$ is

$$\max_{\pi \in \Pi_H} \mathrm{CVaR}_\alpha(Z^\pi(x_0)) \tag{7}$$

We emphasize the importance of the starting state since, unlike the expected value, the CVaR objective is not time-consistent (Pflug and Pichler, 2016). The time inconsistency in this case means that we have to consider the space of all history-dependent policies, and not just the stationary policies (which is sufficient for maximizing e.g. the expected value objective).

## 2.4 Distributional Bellman Operators

The return can be considered not only for the first state $x$, but also for a given first action $a$. Denoting it as $Z(x,a)$, we can define it recursively as follows:

$$\begin{aligned} Z(x,a) &\overset{D}{=} r(x,a) + \gamma Z(x',a') \\ x' &\sim p(\cdot|x,a), a \sim \pi, x_0 = x, a_0 = a \end{aligned} \tag{8}$$

where $\overset{D}{=}$ denotes that random variables on both sides of the equation share the same probability distribution. Analogously to the *policy evaluation* (Sutton and Barto, 1998, p. 90) which estimates the value function $V^\pi$ for a given $\pi$, we speak about value distribution $Z^\pi$.

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

$$\begin{aligned} P^\pi Z(x,a) &\overset{D}{=} Z(x',a') \\ x' &\sim p(\cdot|x,a), a' \sim \pi(\cdot,x) \end{aligned} \tag{9}$$

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

$$\mathcal{T}^{\mathrm{dist}} Z(x,a) \overset{D}{=} r(x,a) + \gamma P^\pi Z(x,a). \tag{10}$$

These operators are described in more detail in (Bellemare et al., 2017).

## 2.5 Value Iteration with CVaR

*Value iteration* (Sutton and Barto, 1998, p. 100) is a RL algorithm for maximizing the expected discounted reward.(Chow et al., 2015) present a dynamic programming formulation for the CVaR MDP problem (7). 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., 2015) proved its convergence.

We repeat their key ideas and results below, as they form a basis for our contributions presented in later sections.

## 2.6 Bellman Equation for CVaR

The results of (Chow et al., 2015) heavily rely on the CVaR decomposition theorem (Pflug and Pichler, 2016):

$$\begin{aligned}&\mathrm{CVaR}_\alpha\left(Z^\pi(x)\right)=\\&\min_{\xi\in\mathcal{U}_{\mathrm{CVaR}}(\alpha,p(\cdot|x,a))}\sum_{x'}p(x'|x,\pi(x))\xi(x')\cdot\\&\qquad\cdot\mathrm{CVaR}_{\xi(x')\alpha}\left(Z^\pi(x')\right)\end{aligned}\tag{11}$$

where the risk envelope $\mathcal{U}_{\mathrm{CVaR}}(\alpha,p(\cdot|x,a))$ coincides with the dual definition of CVaR (6).

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 $\xi$ both appears in 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., 2015) extended the decomposition theorem by defining the *CVaR value function* $C(x,y)$ 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.

$$C(x,y)=\max_{\pi\in\Pi_H}\mathrm{CVaR}_y\left(Z^\pi(x)\right)\tag{12}$$

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

$$\begin{aligned}\mathcal{T}^{\mathrm{cvar}}\mathrm{CVaR}_y(Z(x))=\max_a\Big[&r(x,a)+\\&+\gamma\mathrm{CVaR}_y(P^{\pi^*}Z(x,a))\Big]\end{aligned}\tag{13}$$

where $P^\pi$ denotes the transition operator (9) with an optimal policy $\pi^*$ for all confidence levels.

(Chow et al., 2015, Lemma 3) further showed that the operator $\mathcal{T}^{\mathrm{cvar}}$ is a contraction and also preserves the convexity of $y\mathrm{CVaR}_y$. The optimization problem (11) is a convex one and therefore has a single solution. Additionally, the fixed point of this contraction is the optimal $C^*(x,y)=\max_{\pi\in\Pi}\mathrm{CVaR}_y(Z^\pi(x,y))$ (Chow et al., 2015, Theorem 4).

Naive value iteration with operator $\mathcal{T}^{\mathrm{cvar}}$ is unfortunately unusable in practice, as the state space is continuous in $y$. The approach proposed in (Chow et al., 2015) is then to represent the convex $y\mathrm{CVaR}_y$ as a piece-wise linear function.

## 2.7 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.

$$\begin{aligned}I_x[C](y)=&y_iC(x,y_i)+\\&+\frac{y_{i+1}C(x,y_{i+1})-y_iC(x,y_i)}{y_{i+1}-y_i}(y-y_i)\end{aligned}$$

where $y_i=\max\{y'\in\mathbf{Y}(x):y'\le y\}$. The interpolated Bellman operator $\mathcal{T}^{\mathrm{I}}$ is then also a contraction and has a bounded error ((Chow et al., 2015), Theorem 7).

$$\begin{aligned}\mathcal{T}^{\mathrm{I}}C(x,y)=\max_a\Bigg[&r(x,a)+\\&+\gamma\min_{\xi\in\mathcal{U}_{\mathrm{CVaR}}(\alpha,p(\cdot|x,a))}\sum_{x'}p(x'|x,a)\frac{I_{x'}[C](y\xi(x'))}{y}\Bigg]\end{aligned}\tag{14}$$

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 (14) 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.

# 3 FAST CVaR VALUE ITERATION

We present our original contributions in this section, first describing a connection between the $y\mathrm{CVaR}_y$ 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 $y\mathrm{CVaR}_y$ 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*

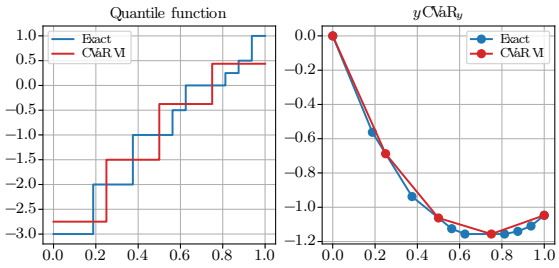


Figure 1: Comparison of a discrete distribution and its approximation according to the CVaR linear interpolation operator.

*$yCVaR_y$ function*

$$yCVaR_y(Z) = \int_0^y VaR_\beta(Z)d\beta \tag{15}$$

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

$$\frac{\partial}{\partial y} yCVaR_y(Z) = VaR_y(Z) \tag{16}$$

### 3.1 CVaR Computation via Quantile Representation

We propose the following procedure: instead of using linear programming for the CVaR computation, we use Lemma 1 and the underlying distributions represented by the $y$CVaR$_y$ function to compute CVaR at each atom. The general steps of the computation are:

1. Transform $y$CVaR$_y(Z(x'))$ of each reachable state $x'$ to a discrete probability distribution using (16).
2. Combine these to to a distribution representing the full state-action distribution
3. Compute $y$CVaR$_y$ for all atoms using (15)

See Figure 2 for a visualization of the procedure. Note that this procedure is linear (in number of transitions and atoms) 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 1 in the bonus materials.

To show the correctness of this approach, we formulate it as a solution to problem (11) in the next paragraphs. Note that we skip the reward and gamma scaling for readability's sake. Extension to the Bellman operator is trivial.

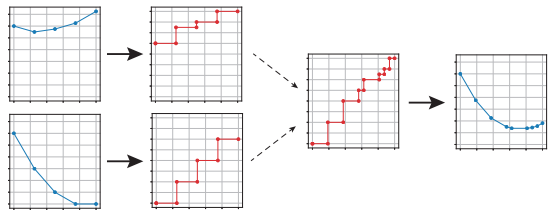

Figure 2: Visualization of the CVaR computation for a single state and action with two transition states. Thick arrows represent the conversion between $y$CVaR$_y$ and the quantile function.

### 3.2 $\xi$-computation

Similarly to Theorem 5 in (Chow et al., 2015), 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: $y_{t+1}$ represents portion of the tail of $Z(x_{t+1})$ that has values present in the computation of CVaR$_{y_t}(Z(x_t))$. In the continuous case, it is the probability in $Z(x_{t+1})$ of values less than VaR$_{y_t}(Z(x_t))$ as we show below.

**Theorem 1.** *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 (11) can be computed for $i = 1, 2$ by setting*

$$\xi(x'_i) = \frac{F_{Z(x'_i)}\left(F^{-1}_{Z(x,a)}(\alpha)\right)}{\alpha} \tag{17}$$

The theorem is straightforwardly extendable to multiple states by induction.

## 4 CVaR Q-LEARNING

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 (Watkins and Dayan, 1992) that works by repeatedly updating the action value estimate according to the sampled rewards and states using a moving exponential average.

As a next contribution, we formulate a Q-learning like algorithm for CVaR.

### 4.1 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.

Given the primal definition of CVaR (3), 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 $\text{VaR}_\alpha$ from data, then use this estimate to compute its $\text{CVaR}_\alpha$. 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, 1951).

The Robbins Monroe theorem has also been applied directly to CVaR estimation by (Bardou et al., 2009), 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 $\text{CVaR}_\alpha$. The procedure requires us to maintain two separate estimates $V$ and $C$, being our VaR and CVaR estimates respectively.

$$V_{t+1} = V_t + \beta_t \left[1 - \frac{1}{\alpha}\mathbb{1}_{(V_t \geq r)}\right] \tag{18}$$

$$C_{t+1} = (1-\beta_t)C_t + \beta_t \left[V_t + \frac{1}{\alpha}(r - V_t)^-\right] \tag{19}$$

$\beta_t$ represents the learning rate at time $t$. An observant reader may recognize a standard equation for quantile estimation in equation (18) (see e.g. (Koenker and Hallock, 2001) 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 (19) then represents the moving exponential average of the primal CVaR definition (3). The estimations are proven to converge, given the usual requirements on the learning rate (Bardou et al., 2009).

### 4.2 CVaR Q-learning

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[2], similar to the definition (12). $V(x,a,y)$ represents the $\text{VaR}_y$ estimate, i.e. the estimate of the $y-$quantile of a distribution recovered from $\text{CVaR}_y$ by Lemma 1.

A key to any temporal difference (TD) 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 1.

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.

The new values $C(x', \bullet)$ 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 1: 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: $\mathbf{d} = \text{extractDistribution}\,(C(x', \bullet), \mathbf{y})$[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: $\quad C(x,a,y_i) = (1-\beta)C(x,a,y_i) + \beta\left[V(x,a,y_i) + \frac{1}{y_i}(r + \gamma d_j - V(x,a,y_i))^-\right]$
9: **end for**

If the atoms aren't uniformly spaced (log-spaced atoms are motivated by the error bounds of CVaR Value Iteration), we have to perform basic importance

[2] We can read $(x,y)$ as extended state - combining the information about (i) environment and (ii) risk perception. In this sense, the extension is similar to extended reinforcement Q learning as described in (Obayashi et al., 2015) where the new component expresses the expresses the emotional state of the agent.

[3] Extracts the underlying distribution from CVaR, see Algorithm 1 in bonus materials.

sampling when updating the estimates . 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

$$V(x,a,y_i) = V(x,a,y_i) + \beta \mathop{\mathbb{E}}_j \left[ 1 - \frac{1}{y_i} \mathbb{1}_{(V(x,a,y_i) \geq r + \gamma d_j)} \right]$$
$$C(x,a,y_i) = (1-\beta) C(x,a,y_i) + \\ + \beta \mathop{\mathbb{E}}_j \left[ V(x,a,y_i) + \frac{1}{y_i} (r + \gamma d_j - V(x,a,y_i))^- \right] \quad (20)$$

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.

## 4.3 VaR-based Policy Improvement

CVaR Q-learning helps us to find the $\text{CVaR}_y$ function, but does not help us with retrieving the optimal policy. Below we formulate an algorithm that allows us to get the optimal policy.

Let us 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 $\text{CVaR}_\alpha$ criterion for selected $\alpha$.

Recall the primal definition of CVaR (3)

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

Our goal (7) 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

$$\max_\pi \text{CVaR}_\alpha(Z^\pi) = \max_\pi \frac{1}{\alpha} \mathbb{E} \left[ (Z^\pi - s^*)^- \right] + s^* \quad (21)$$

Given that we have access to the return distributions, we can improve the policy by simply choosing an action that maximizes $\text{CVaR}_\alpha$ 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 $\pi$ 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. $\pi$ while keeping $s$ fixed. This is the policy improvement step.
3. Recompute $\text{CVaR}_\alpha(Z^{\pi^*})$ where $\pi^*$ 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 2 for the full procedure which we justify in the following theorem.

Algorithm 2: VaR-based policy improvement.

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

**Theorem 2.** *Let $\pi$ be a stationary policy, $\alpha \in (0, 1]$. By following policy $\pi^*$ from algorithm 2, we improve $CVaR_\alpha(Z)$ in expectation:*

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

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, 2011) although not to this extent. See Remark 3.9 in (Bäuerle and Ott, 2011) for details.

### 4.3.1 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 $\mathbb{E}[(Z_t - s)^-]$ 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.

To solve this, we propose an approximate heuristic that uses linear interpolation to extract the VaR of given distribution.

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

See Algorithm 2 and Figure 1 in the bonus materials for more intuition behind the heuristic.

# 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. (Mnih et al., 2015) 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 section, we extend CVaR Q-learning to its deep Q-learning variant and show the practicality and scalability of the proposed methods.

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. First significant change compared to DQN or QR-DQN (Dabney et al., 2017) 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.

## 5.1 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 $\mathcal{T}V = r + \gamma\mathbf{d}$.

$$\mathcal{L}_{\text{VaR}} = \sum_{i=1}^{N} \mathbb{E}_j \left[ (r + \gamma d_j - V_i(x,a))(y_j - \mathbb{1}_{(V_i(x,a) \geq r + \gamma d_j)}) \right] \tag{22}$$

where $d_j$ are atoms of the extracted distribution.

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

$$\mathcal{L}_{\text{CVaR}} = \sum_{i=1}^{N} \mathbb{E}_j \left[ \left( V_i(x,a) + \right. \right. \\ \left. \left. + \frac{1}{y_i} (r + \gamma d_j - V_i(x,a))^- - C_i(x,a) \right)^2 \right] \tag{23}$$

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

$$\mathcal{L} = \mathcal{L}_{\text{VaR}} + \mathcal{L}_{\text{CVaR}} \tag{24}$$

Combining the loss functions with the full DQN algorithm, we get the full CVaR-DQN with experience replay[4]. Note that we utilize a target network $C'$ that is used for extraction of the target values of $C$, similarly to the original DQN. The network $V$ does not need a target network since the target is constructed independently of the value $V$.

# 6 EXPERIMENTS[5]

## 6.1 CVaR Value Iteration

We test the proposed algorithm on the same task as (Chow et al., 2015). 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 $40 \times 60$ grid-world and approximate the $\alpha\text{CVaR}_\alpha$ function using 21 log-spaced atoms. The learned policies on a sample grid are shown in Figure 3.

[4] See Algorithm 3 in the bonus materials.

[5] All code is publicly available at https://bit.ly/2YFCDyE

While (Chow et al., 2015) report computation time on the order of hours (using the highly optimized CPLEX Optimizer), our naive Python implementation converged under 20 minutes.

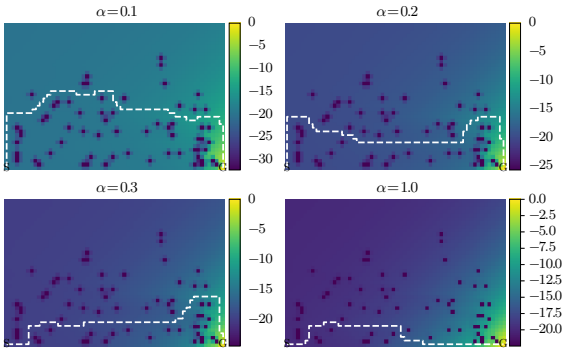


Figure 3: Grid-world simulations. The optimal deterministic paths are shown together with CVaR estimates for given α.

## 6.2 CVaR Q-learning

We use the same gridworld for our experiments. Since the positive reward is very sparse, we chose to run CVaR Q-learning on a smaller environment of size $10 \times 15$. 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 $\varepsilon$. We found that lower $\varepsilon$ values led to overfitting the optimal expected value policy as the agent updated states out of the optimal path sparsely.

With said parameters, the agent was able to learn the optimal policies for different levels of α. See Figure 4 for learned policies and Figure 5 for Monte Carlo comparisons.

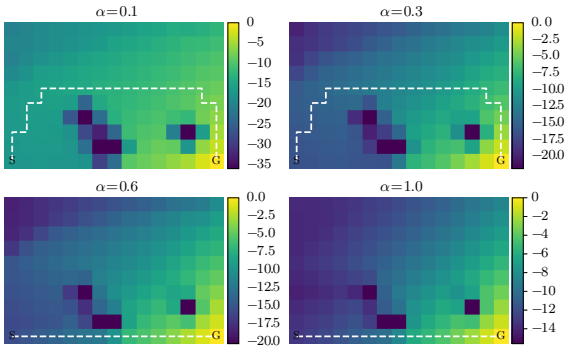


Figure 4: Grid-world Q-learning simulations. The optimal deterministic paths are shown together with CVaR estimates for given α.

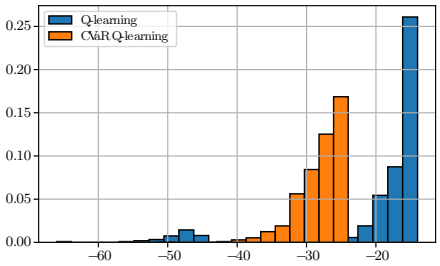


Figure 5: Histograms from 10000 runs generated by Q-learning and CVaR Q-learning with $\alpha = 0.1$.

## 6.3 Deep CVaR Q-learning

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.

### 6.3.1 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 6.

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.

### 6.3.2 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.

The architecture used in our experiments differs slightly from the original one used in DQN. In our

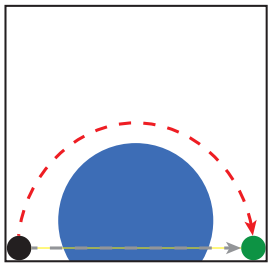

Figure 6: The Ice Lake environment. The agent is black and his target is green. The blue ring represents a dangerous area with risk of breaking the ice. Grey arrow shows the optimal risk-neutral path, red shows the risk-averse path.

case the output is not a single value but instead a vector of values for each action, representing $\text{CVaR}_y$ or $\text{VaR}_y$ 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 (Kingma and Ba, 2014) as the optimization algorithm.

The implementation was done in Python and the neural networks were built using Tensorflow (Abadi et al., 2016) as the framework of choice for gradient descent. The code was based on OpenAi baselines (Dhariwal et al., 2017), an open-source DQN implementation.

#### 6.3.3 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 $\varepsilon$-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 $\alpha$. 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 $\varepsilon$ than DQN. 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 $\alpha$ 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.

#### 6.3.4 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, as in Figure 6[6].

Although we tested with the vanilla version of DQN, we expect that all the DQN improvements such as experience replay (Hessel et al., 2017), dueling (Wang et al., 2015), parameter noise (Plappert et al., 2017) and others (combining the improvements matters, see (Hessel et al., 2017)) should have a positive effect on the learning performance. Another practical improvement may be the introduction of Huber loss, similarly to QR-DQN.

## 7 CONCLUSION

In this paper, we tackled the problem of dynamic risk-averse reinforcement learning. Specifically we focused on optimizing the Conditional Value-at-Risk objective.

The work mainly builds on the CVaR Value Iteration algorithm (Chow et al., 2015), 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

[6]See videos in the bonus materials.

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 correctness 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 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.

## 7.1 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.

We also 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, 2011) 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 (Bellemare et al., 2017) and QR-DQN (Dabney et al., 2017). 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 apply the techniques on relevant problems from the financial sector and on practical robotics, and other risk-sensitive applications, including interdisciplinary reasearch on emotional perception of risk (Obayashi et al., 2015).

# REFERENCES


Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bardou, O., Frikha, N., and Pages, G. (2009). Recursive computation of value-at-risk and conditional value-at-risk using mc and qmc. In *Monte Carlo and quasi-Monte Carlo methods 2008*, pages 193–208. Springer.

Bäuerle, N. and Ott, J. (2011). Markov decision processes with average-value-at-risk criteria. *Mathematical Methods of Operations Research*, 74(3):361–379.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*.

Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684.

Chow, Y., Tamar, A., Mannor, S., and Pavone, M. (2015). Risk-sensitive and robust decision-making: a cvar optimization approach. In *Advances in Neural Information Processing Systems*, pages 1522–1530.

Committee, B. et al. (2013). Fundamental review of the trading book: A revised market risk framework. *Consultative Document, October*.

Coraluppi, S. P. (1998). Optimal control of markov decision processes for performance and robustness.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018). Implicit quantile networks for distributional reinforcement learning.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2017). Distributional reinforcement learning with quantile regression. *arXiv preprint arXiv:1710.10044*.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2017). Openai baselines. https://github.com/openai/baselines.

Garcıa, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480.

Hamid, O. and Braun, J. (2019). *Reinforcement Learning and Attractor Neural Network Models of Associative Learning*, pages 327–349.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*.

Howard, R. A. and Matheson, J. E. (1972). Risk-sensitive markov decision processes. *Management science*, 18(7):356–369.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koenker, R. and Hallock, K. F. (2001). Quantile regression. *Journal of economic perspectives*, 15(4):143–156.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Leike, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., Orseau, L., and Legg, S. (2017). Ai safety gridworlds. *arXiv preprint arXiv:1711.09883*.

Lesser, K. and Abate, A. (2017). Multi-objective optimal control with safety as a priority. In *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*, pages 25–36.

Macek, K. (2010). Predictive control via lazy learning and stochastic optimization. In *Doktorandské dny 2010 - Sborník doktorandů FJFI*, pages 115–122.

Majumdar, A. and Pavone, M. (2017). How should a robot assess risk? towards an axiomatic theory of risk in robotics. *arXiv preprint arXiv:1710.11040*.

Miller, C. W. and Yang, I. (2017). Optimal control of conditional value-at-risk in continuous time. *SIAM Journal on Control and Optimization*, 55(2):856–884.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Morimura, T., Sugiyama, M., Kashima, H., Hachiya, H., and Tanaka, T. (2012). Parametric return density estimation for reinforcement learning. *arXiv preprint arXiv:1203.3497*.

Obayashi, M., Uto, S., Kuremoto, T., Mabu, S., and Kobayashi, K. (2015). An extended q learning system with emotion state to make up an agent with individuality. *2015 7th International Joint Conference on Computational Intelligence (IJCCI)*, 3:70–78.

Pflug, G. C. and Pichler, A. (2016). Time-consistent decisions and temporal decomposition of coherent risk functionals. *Mathematics of Operations Research*, 41(2):682–699.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2017). Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

Prashanth, L. (2014). Policy gradients for cvar-constrained mdps. In *International Conference on Algorithmic Learning Theory*, pages 155–169. Springer.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Rockafellar, R. T. and Uryasev, S. (2000). Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

Sobel, M. J. (1982). The variance of discounted markov decision processes. *Journal of Applied Probability*, 19(4):794–802.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Tamar, A., Chow, Y., Ghavamzadeh, M., and Mannor, S. (2017). Sequential decision making with coherent risk. *IEEE Transactions on Automatic Control*, 62(7):3323–3338.

Tamar, A., Glassner, Y., and Mannor, S. (2015). Optimizing the cvar via sampling. In *AAAI*, pages 2993–2999.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.