DEPARTMENT OF
STATISTICS

General Value Distribution

by

Louis Serrano

Wolfson College

A dissertation submitted in partial fulfilment of the degree of Master of Science in Statistical Science.

Department of Statistics, 24–29 St Giles,
Oxford, OX1 3LB

September 2020

This is my own work (except where otherwise indicated)

Candidate: Full Name

Signed:.....

Date:.....

Abstract

The contribution of this Dissertation is two-fold. First, we combine ideas of General Value Function (GVF) with methods from Distributional Reinforcement Learning (DRL) to form the General Value Distribution (GVD) framework . General Value function allows an agent or a horde - an agent with several sub-agents - to learn a larger set of value functions, while DRL estimates the distribution of the returns, rather than just the expectation. We focus our work on Quantile General Value Distribution (QGVD), the GVD version of Quantile Distributional Reinforcement Learning (QDRL). We adapt from DRL theoretical guarantees for GVD and QGVD. Using QGVD, we then provide two new algorithms: the Average Loss Policy Evaluation and Risk-Sensitive Q-Learning. The first one is a new method for Policy Evaluation and works by averaging the quantile huber loss over all possible actions. The second one builds on utility theory to provide a risk-sensitive policy improvement scheme. We show experimental mean and distributional convergence for the Average Loss algorithm and show coherent behaviour for the risk-sensitive Q-Learning. We also present a Unified Horde architecture that can learn multiple GVDs with only one Neural Network. Second, we present a naive Monte Carlo approach to estimate the Kernel Mean Embedding of the returns distribution. We show that we can retrieve approximately the q values using this embedding.

Acknowledgements

First and foremost, I would like to thank my academic supervisor Prof. Yee Whye Teh. This challenging thesis would not have been possible without him. He gave me the freedom to approach the project the way I wanted and often put me back on track with his precious advice.

I also wish to deeply thank my parents for their unconditional support throughout my education and especially for their assistance during this pandemic. They have accommodated me for the second part of this MSc and have put me in the best conditions to take the exams. I wish them good luck for their new life in Singapore and I am eternally grateful for all their help.

Acronyms

CDRL Categorical Distributional Reinforcement Learning. 12

DRL Distributional Reinforcement Learning. 1

GVD General Value Distribution. 2

GVF General Value Function. 1

QDRL Quantile Distributional Reinforcement Learning. 13

QGVD Quantile General Value Distribution. 23

RKHS Reproducing Kernel Hilbert Space. 2

RL Reinforcement Learning. 1

RMSVE Root Mean Square Value Error. 39

TD Temporal Difference. 5

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Objectives

Classical Reinforcement Learning (RL) aims to evaluate and improve a given policy by adopting a so-called "expected" perspective. Among the different objectives, the core idea is usually to consider a value function, defined as the expected return that an agent will get given a state or state-action pair and following a given policy. The goal in conventional Reinforcement Learning is to find a policy that maximises this value function.

The GVF framework introduced in 2011 by Sutton et al [26]., extends this expected paradigm to learn a cumulant-based target that we do not necessarily wish to maximise but only to predict. The elaboration of GVF was in fact motivated to efficiently represent an agent's predictive knowledge of its own environment. By learning multiple GVFs in parallel the agent broadens the representation of its environment.

Bellemare et al. [2], proposed in 2017 a distributional approach to RL by taking into account the full distribution of the returns, instead of looking only at their expectations. Several algorithms have been developed to approximate the distribution of the returns. DRL has shown state-of-the-art performances in the Atari-2600 benchmarks with the C51 and QR-DQN algorithms and is starting to attract a lot of research attention.

On the other hand, kernel methods have been widely used in Machine Learning for the past 30 years. They are used to incorporate non-linearities in supervised or unsupervised learning methods. Famous examples include Kernel PCA, Kernel SVM, Kernel Ridge Regression, Gaussian Process, etc. Kernels express a notion of dissimilarity and can be applied to any kind of data, as long as one can find a positive definite function that makes sense. In 2007, Smola et al. [22] presented the idea of Kernel Mean Embedding that maps

probability distributions to functions in a Reproducing Kernel Hilbert Space (RKHS).

The objectives of this thesis are first to combine methods from Distributional Reinforcement Learning with ideas from General Value Function to have an equivalent of GVF but with distributional techniques. We also want to explore the possibility of deriving a policy evaluation algorithm based on the Kernel Mean Embedding representation of the conditional returns.

1.2 Document structure

In Chapter 2, we mention important results from classical Reinforcement Learning, General Value Function, Distributional Reinforcement Learning and Kernel Mean Embedding that we will need for the rest of the dissertation.

In Chapter 3, we propose the GVD framework, which combines ideas from GVF and DRL. We focus on the quantile approximation of DRL.

In Chapter 4, we derive the Naive Monte Carlo Mean Embedding algorithm, an on-policy tool to learn the conditional mean embedding of the returns.

In Chapter 5, we present the experiments that were tried with methods exposed in Chapter 3 and Chapter 4.

Chapter 2

Background

This chapter aims to provide the reader with a brief introduction of the mathematical and algorithmic frameworks that will be discussed throughout the thesis. We will first expose key notions of Reinforcement Learning before moving on to the GVF framework. Then we will tackle the distributional perspective of RL, and finally we will recall important results for Kernel and Mean Embedding.

2.1 Reinforcement Learning

This section is largely inspired by the Introduction to Reinforcement Learning from Sutton and Barto [25].

We will consider an **agent** that interacts with an **environment** as follows:

1. At time step t, the agent is in a **state** S_t.
2. The agent takes an **action** A_t conditionally on S_t, according to a policy π:

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

3. According to the environment dynamics, it ends up in a new state S_{t+1} and receives a **reward** R_{t+1} along the way.
4. If the task is **episodic**, we denote T the time step at which the agent reaches a terminal state. For **continuing** tasks, $T = \infty$.

Formally, this interaction can be described by a time-homogeneous **Markov Decision Process** (MDP): $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are respectively the state and action spaces. R is the reward function, P is the transition kernel : $P(S_{t+1}, R_{t+1}|S_t, A_t)$ and γ

is the discount factor in $[0, 1]$. The action space is finite: $\forall t \in \{1 : T-1\}, A_t \in \{1, ..., k\}$ with $k \in \mathbb{N}$.

The **return** is defined as $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$. Note that this is a power series of γ, and thus if $T < \infty$ or if $\gamma < 1$, then the return is finite if the sequence of rewards is bounded. The **value function** of a state is the expectation of the return, given the agent starts in this state and then follows a given policy π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}[G_t|S_t = s, A_{t:T-1} \sim \pi] \tag{2.1}$$

The **state-action value function**, or simply action value function, is obtained by assigning a specific next action:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}[G_t|S_t = s, A_t = a, A_{t+1:T-1} \sim \pi] \tag{2.2}$$

Using the fact that we can write $G_t = R_{t+1} + \gamma G_{t+1}$, we obtain the **Bellman equations** for the state and action value functions:

$$v_\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)} \mathbb{E}_{(r,s')\sim P(\cdot|s,a)}[r + \gamma v_\pi(s')] \tag{2.3}$$

$$q_\pi(s, a) = \mathbb{E}_{(r,s')\sim P(\cdot|s,a)} \mathbb{E}_{a'\sim\pi(\cdot|s')}[r + \gamma q_\pi(s', a')] \tag{2.4}$$

For finite MDPs, the value function induces a partial ordering of policies ([25] section 3.6). We say that $\pi \leq \pi'$ if and only if $\forall s \in \mathcal{S}, v_\pi(s) \leq v_{\pi'}(s)$. A policy that is better than or equal to all the other policies is called **optimal** and is noted π_* even though it might not be unique. The optimal state value function is evaluated at an optimal policy: $v_*(s) = \max_\pi v_\pi(s)$ and the same goes for the optimal state-action value function: $q_*(s, a) = \max_\pi q_\pi(s, a)$.

Since,

$$v_\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)} q_\pi(s, a) \tag{2.5}$$

we derive the **Bellman optimality equations** for v_* and q_*.

$$v_*(s) = \max_a q_{\pi^*}(s, a) = \max_a \mathbb{E}_{(r,s')\sim P(\cdot|s,a)}[r + \gamma v_*(s)] \tag{2.6}$$

$$q_*(s, a) = \mathbb{E}_{(r,s')\sim P(\cdot|s,a)}[r + \max_{a'} q_*(s', a')] \tag{2.7}$$

There are two main separate objectives in the RL literature: **policy evaluation** and **policy improvement**. The first one, also called the prediction problem is to find an estimated

state or state-action value function for a fixed policy. The latter consists of improving a current policy, so that the new policy is better than the previous one using the value function ordering. Several methods exist in the literature to achieve these objectives.

For Policy evaluation:

- If we know the dynamics of the model, we can apply **Dynamic Programming** (DP) to iteratively update the state values using the value function Bellman equation. In practice, we rarely know the dynamics of the environment, which makes DP impracticable.
- With unknown environment dynamics, we can still find the state values by sampling transitions and observing returns according to our policy. For instance, if the task is episodic, first-visit **Monte Carlo** estimates the value function by averaging returns: $v(s) = \frac{1}{N}\sum_{n=1}^{N} G^{(n)}(s)$ where $G^{(n)}(s)$ is the return obtained after the first visit in s in the n-th episode. MC methods update the values only at the end of an episode, which can lead to high variance and can result in slow learning. On the other hand, **Temporal Difference (TD)** uses the following update rule after each transition: $v(s) \leftarrow v(s) + \alpha[R_{t+1} + \gamma v(S_{t+1}) - v(s)]$. α is the step-size parameter and is in [0,1]. The main advantage of TD is that the agent need not wait until the end of an episode for learning to occur. It can be done in an online fashion, even with continuing tasks.

For Policy iteration, we focus on methods that estimate state-action value functions, for which we can distinguish **on-policy** from **off-policy** techniques:

- **SARSA** is an on-policy TD algorithm for the state-action value function, whose updates are made after each sampled transition from the agent's policy: $q(s,a) \leftarrow q(s,a) + \alpha[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(s,a)]$. It is on-policy as the next action A_{t+1} is obtained by sampling from the policy π. The target policy is greedy to some extent to the estimated action values.
- **Q-Learning** is an off-policy method. The updates are done as follows: $q(s,a) \leftarrow q(s,a) + \alpha[R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(s,a)]$. Off-policy methods explore the state-action space with a behaviour policy b, but updates for control are done with a different target policy π. b can be ϵ-greedy policy with respect to q while π is the greedy policy. One constraint is to choose a behaviour policy that continues exploring the state space. This implies that we can find deterministic policy while continuing exploration.

There are also on-policy and off-policy algorithms for policy evaluation. Tabular meth-

ods, such as the ones presented above, work by storing estimated state and action values for each state and each state-action pair. Naturally, this is infeasible if the state space is infinite. Nevertheless, we can **approximate** these values by using a representative feature $\boldsymbol{x}$ and a weight vector $\boldsymbol{w}$ to form parametric estimates, $\hat{v}(s, \boldsymbol{w}) \simeq v_\pi(s)$ and $\hat{q}(s, a, \boldsymbol{w}) \simeq q_\pi(s, a)$. The idea is then to use supervised learning tools to learn the weights that minimize a loss-type objective.

There is a wide body of literature that covers the properties of linear approximation, where each value and action value are respectively estimated by $\boldsymbol{w}^T\boldsymbol{x}(s)$ and $\boldsymbol{w}^T\boldsymbol{x}(s, a)$. However, we can use any type of function to represent them, even decision trees. Deep Learning methods have been gaining a lot of momentum in the RL community in the past decade, with groundbreaking results such as AlphaGo and popular Q-Learning methods such as DQN (Mnih et al., 2015 [16]).

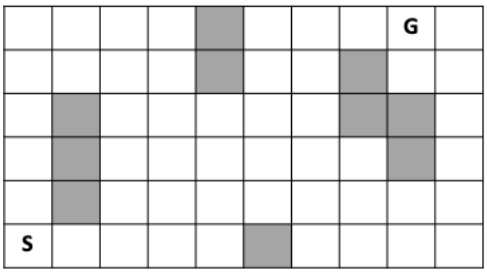
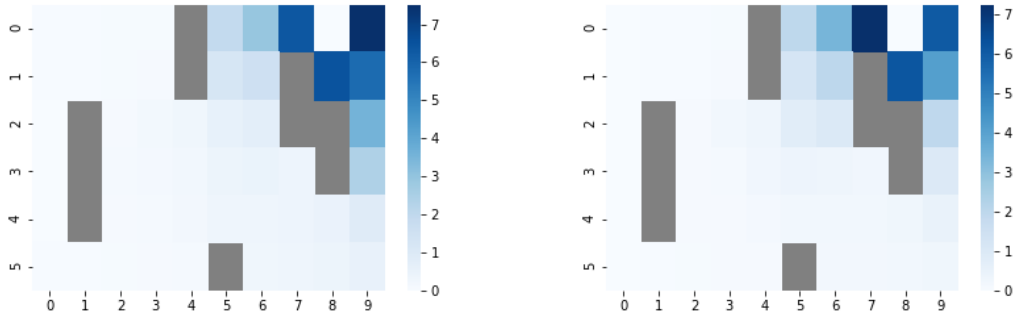


Figure 2.1: The Maze environment used for the Background experiments. In grey, the obstacles.

For the sake of concreteness, we now consider an agent that can take 4 actions (up, right, down, left) in a maze environment with obstacles (see Figure 2.1). The agent starts in S. The state transitions are deterministic, unless the action leads to an obstacle or outside the maze, in which case the agent stays in the same state. On each transition, the agent gets a reward of zero unless it arrives to the G square where it gets a reward of 10. We use a discount factor γ of 0.95.



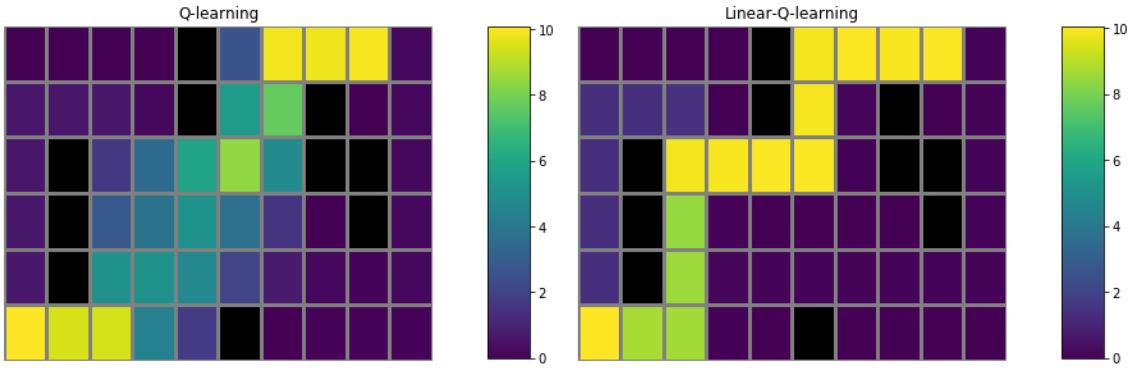
(a) TD agent, $\alpha = 0.1$ (b) TD agent with tile coding, $\alpha = 0.1/4$

Figure 2.2: State values over the Maze environment obtained after 100 episodes. The value of the terminal state is set to zero.

We first begin to evaluate the random policy that selects an action with probability 0.25, and then use Q-Learning to find an optimal policy. Both tasks are solved for the tabular case and with linear approximation, whose features were constructed with 8 tilings and 4 tiles (see Chapter 9 of [25]).

Figure 2.2 shows the state values across the maze for the random policy after 100 episodes. We can observe that in both cases, the algorithms give higher values to state closer to the goal state, and that both solutions are somewhat similar.

For the control setting, we use an ϵ-greedy policy for the behaviour policy with an ϵ value of 0.01. We plot in Figure 2.3 the average number of visits per state during the last 10 episodes of a 500-episode run. We averaged the results over 100 runs. We notice that the agent in both cases learns effectively to reach the target through one of the shortest path. According to Figure 2.4, the Q-Learning agents converge quickly to their optimal policies as the number of steps per episode decays before levelling of after 25 episodes.



(a) Q-learning agent, $\alpha = 0.1$, $\epsilon = 0.01$

(b) Q-learning agent with tile coding, $\alpha = 0.1/4$, $\epsilon = 0.01$

Figure 2.3: Average number of state visits during the last 10 episodes of a 500-episode run

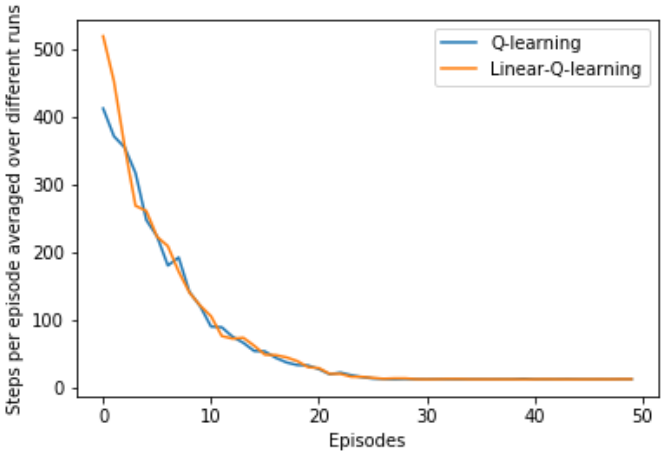


Figure 2.4: Average steps per episode throughout a run.

2.2 General Value Function

The key idea behind GVF is to extend the purpose of the value function to a broader and more general representation of **predictive knowledge** of the world, thereby leveraging the strengths of conventional value function learning methods (White 2015 [27]). We expose here the setting of GVFs as described in Chapter 4 of White's thesis.

The setup is similar to that introduced in Section 2.1, but the states are not necessarily observable. Instead, the agent has access to features $\boldsymbol{x}_t \in \mathbb{R}^n$ and takes actions A_t sequentially: $(\boldsymbol{x}_t, A_t)$, $(\boldsymbol{x}_{t+1}, A_{t+1})$, $(\boldsymbol{x}_{t+2}, A_{t+2})$, etc. The objective being predictive, the learning amounts to estimate a target scalar signal $\tilde{G}_t \in \mathbb{R}$.

The target is computed from two signals, the **cumulant** C_t that plays a similar role to the reward of Section 2.1 and the **termination signal** $\gamma_t \in [0, 1]$. Using the convention that $\prod_{j=1}^{0} \gamma_{t+j} = \gamma^0 = 1$, we express the target

$$\tilde{G}_t = \sum_{k=0}^{\infty} (\prod_{j=1}^{k} \gamma_{t+j}) C_{k+t+1} \tag{2.8}$$

Note here that the main difference with the previous RL return is that the termination signal is a non-constant discount factor. Formally γ_t, C_t and $\boldsymbol{x}_t$ are the outputs of a termination function $\gamma : \mathcal{S} \to [0, 1]$, a cumulant function $c : \mathcal{S} \to \mathbb{R}$ and a feature function $\boldsymbol{x} : \mathcal{S} \to \mathbb{R}^n$ that are defined over the state space of an unobservable MDP. Similarly, the actions are taken according to a behaviour policy $b : \mathcal{S} \times \mathcal{A} \to [0, 1]$, which maps a state-action pair to a probability mass.

The term cumulant is used instead of reward because we do not necessarily wish to maximise the amount of cumulant the agent receives, which is usually the case in RL. Here, the goal is to accurately predict the future sum of cumulants. The cumulant can be any kind of scalar, that could enrich the agent's knowledge of its environment.

For episodic tasks, the discount factor can be 1, unless we wish to enforce that distant rewards are worth less than more immediate rewards. For continuing tasks, we choose a constant $\gamma < 1$ to have a finite return. We can recover these two cases with the GVF framework. For episodic tasks, simply set γ_t to a constant until the agent reaches a terminal state for which we set $\gamma(S_T) = 0$. If the problem is continuing, simply set γ_t to a constant less than one.

We can think of the γ function as the probability of continuing the current trajectory when arriving in a certain state. In an episodic case, setting $\gamma(S_T)$ to 0 means the agent considers the cumulants up to T and is equivalent to giving a probability of zero for continuing the trajectory. Note that the termination probabilities given by $1 - \gamma_t$ are hypothetical since they need not be associated with an actual interruption of the trajectory. Those are given by the behaviour policy.

A state GVF takes as inputs the three *question functions* that are the target policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, the termination signal $\gamma : \mathcal{S} \to [0, 1]$, and the cumulant function $c : \mathcal{S} \to \mathbb{R}$ and is defined as a conditional expectation like in (2.1):

$$v(s; \pi, \gamma, c) = \mathbb{E}_\pi[\tilde{G}_t | S_t = s] \tag{2.9}$$

Likewise, we can define a state-action GVF:

$$q(s, a; \pi, \gamma, c) = \mathbb{E}_\pi[\tilde{G}_t | S_t = s, A_t = a] \tag{2.10}$$

The GVF tries to answer the following question: What is the expected value of the target if we follow the policy π, given these cumulant and termination signals ?

To learn an approximate answer to that question, we could use any kind of algorithm from the RL literature that can learn value and state action value functions. The only difference is that γ is not constant anymore.

[27] uses linear function approximation for state and action values and therefore adopts algorithms that are suited for this setting, such as

- TD(λ), an on-policy method to learn state values introduced by Modayil, White & Sutton in 2014
- GTD(λ), an off-policy method to learn state values introduced by Maei in 2011
- GQ(λ), an off-policy method to learn state-action values introduced by Maei & Sutton in 2010 [14]

In these methods, the three *answer functions* that specify the learning approximation are the feature vector generation scheme $\boldsymbol{x} : \mathcal{S} \to \mathbb{R}^n$, the eligibility trace-decay rate $\lambda : \mathcal{S} \to [0, 1]$, and the behaviour policy b.

In conventional RL, the **trace-decay** parameter λ is constant and changes the target fed into the TD, SARSA or Q-learning algorithms. The **$\boldsymbol{\lambda}$-return** G_t^λ is defined as a weighted average of all n-step returns $G_{t:t+n}$.

$$G_{t:t+n} = \sum_{k=t+1}^{t+n} \gamma^{k-t-1} C_k + \gamma^n \hat{v}(S_{t+n}, \boldsymbol{w}_{t+n-1}) \tag{2.11}$$

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \tag{2.12}$$

In (2.12), notice that if $\lambda = 0$, we get $G_t^0 = G_{t:t+1} = C_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}_t)$ which is the objective used in the Temporal Difference update. It can be shown that for episodic tasks, the choice $\lambda = 1$ yields the objective used with Monte Carlo techniques.

The GVF was actually introduced in 2011 by Sutton et al. [26] as part of the Horde architecture that consists of a main agent composed of many sub-agents, called **demons**. In this architecture, each demon is asked a different question about the environment through

the question functions and handles therefore a specific GVF. A demon can learn its GVF on-policy or off-policy and all demons learn in parallel. Off-policy demons are more adapted to learn from a sequence of actions generated by a behaviour policy than on-policy demons, because their policy would rarely match the behaviour policy [26].

2.3 Distributional Reinforcement Learning

We use the MDP setting as described in Section 2.1. We define the return Z_π as the random variable whose expectation is taken to compute the value of a state-action pair. Formally,

$$Z_\pi(s,a) \sim G_t | S_t = s, A_t = a, A_{t:T-1} \sim \pi \tag{2.13}$$

so that

$$q_\pi(s,a) = \mathbb{E}[Z_\pi(s,a)] \tag{2.14}$$

One can see Z_π as a mapping from state-action pairs to distributions over returns and we will refer to it as the *value distribution* [2]. Z_π satisfies a distributional Bellman equation analogous to Equation (2.2)

$$\begin{aligned} Z_\pi(s,a) &\overset{D}{=} \mathcal{T}^\pi Z_\pi(s,a) \\ &\overset{D}{=} R(s,a) + \gamma Z_\pi(S', A') \end{aligned} \tag{2.15}$$

where $X \overset{D}{=} Y$ means that the r.v. X and Y have the same probability laws, $(S', R) \sim P(\cdot|s,a)$, $A' \sim \pi(\cdot|S')$ and $\mathcal{T}^\pi$ is the distributional Bellman operator. Similarly, the distributional optimality Bellman equation is given by

$$\begin{aligned} Z(s,a) &\overset{D}{=} \mathcal{T} Z(s,a) \\ &\overset{D}{=} R(s,a) + \gamma Z(S', \arg\max_{a' \in \mathcal{A}} \mathbb{E} Z(S', a')) \end{aligned} \tag{2.16}$$

where $\mathcal{T}$ is the distributional optimality Bellman operator. Note that in (2.16), the action taken by $\mathcal{T}$ maximizes the expected value of the return, as in conventional RL. Before presenting the ideas used in the literature to perform policy evaluation and control, let us first recall some notions of distance in the space of distributions.

Let $\mathcal{P}(\mathbb{R})$ be the set of all probabillity measures on $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$. Let $P, Q \in \mathcal{P}(\mathbb{R})$. The **Kullback-Leibler** divergence from P to Q is

$$D_{KL}(P||Q) = \mathbb{E}_{X \sim P} \log \frac{P(X)}{Q(X)} \tag{2.17}$$

The KL divergence is not a metric as it is not symmetric. However, it is non-negative (Gibbs' inequality) and $D_{KL}(P||Q) = 0$ iff $Q = P$ a.s. . The KL divergence has interesting properties but its value diverges when P is not absolute continuous with respect to Q.

Let $(\mathcal{X}, \|\cdot\|)$ be a Polish space (complete separable metric space) and let $\mathcal{P}_p(\mathcal{X})$ be the set of probability measures with finite pth moments, with $p \geq 1$. Let $\mu, \nu \in \mathcal{P}_p(\mathcal{X})$ and M, N be their respective cumulative distribution functions. The **p-Wasserstein** metric, between μ and ν is

$$\begin{aligned} d_p(\mu, \nu) &= (\inf_{(U,V) \in \Pi_{\mu\nu}} \mathbb{E}\|U - V\|^p)^{1/p} \\ &= (\mathbb{E}\left\|M^{-1}(\mathcal{U}) - N^{-1}(\mathcal{U})\right\|^p)^{1/p} \end{aligned} \tag{2.18}$$

where $\Pi_{\mu\nu}$ is the space of pairs of random variables with marginal probabilities μ and ν and $\mathcal{U} \sim \text{Uniform}([0, 1])$. We write $d_p(U, V) = d_p(f_U, f_V)$ where f_U and f_V are the marginal densities. The Wasserstein metric is particularly interesting as it does not suffer from disjoint support, however it cannot be minimized using stochastic gradient descent [5]. Let $\mathcal{Z}$ denote the space of value distributions with bounded moments. For two value distributions $Z_1, Z_2 \in \mathcal{Z}$ we use the maximal form of Wasserstein metric: $\bar{d}_p = \sup_{s,a} d_p(Z_1(s, a), Z_2(s, a))$.

Noting F_P and F_Q the cumulative distribution functions of P and Q, the **Cramér** distance between P and Q is

$$D_C(P, Q) = \int_{-\infty}^{\infty} (F_P(t) - F_Q(t))^2 dt \tag{2.19}$$

Note that unlike KL divergence, the Cramér distance defines a proper distance.

Since the space of distributions $\mathcal{P}(\mathbb{R})$ is infinite-dimensional, it is impossible to work directly with the distributional Bellman equation, and current approaches to distributional RL generally rely on parametric approximations to this equation [20].

[2] introduced the C51 algorithm which is based on Categorical Distributional RL. They showed that the distributional Bellman operator $\mathcal{T}^\pi$ is a γ-contraction with respect to $\bar{d}_p$, which guarantees convergence results for policy evaluation. Policy iteration was demonstrated to be less stable as the distributional optimality Bellman operator $\mathcal{T}$ is not a contraction in any metric over distributions.

Categorical Distributional Reinforcement Learning (CDRL) takes as inputs V_{MIN}, V_{MAX}, and Δz and constructs a fixed support of points $z_i = V_{\text{MIN}} + i\Delta z$ for $0 \leq i \leq N$. It then ap-

proximates the distribution over returns by assigning variable probabilities $\mathbb{P}(Z_\theta(s,a) = z_i) = p_i = \frac{e^{\theta_i(s,a)}}{\sum_j e^{\theta_j(s,a)}}$. Since we cannot minimize the Wasserstein distance with stochastic gradient descent, the authors resorted to minimize the KL divergence between the current distribution of $Z(s,a)$ and the projected sample Bellman update $\hat{\mathcal{T}}Z(s,a)$.

The heuristic projection step guarantees that the two distributions do not have disjoint supports and is defined as an application that maps a dirac distribution to a combination of diracs over the fixed support

$$\Pi_C(\delta_y) = \begin{cases} \delta_{z_1} & \text{if } y \leq z_1 \\ \frac{z_{i+1}-y}{z_{i+1}-z_i}\delta_{z_i} + \frac{y-z_i}{z_{i+1}-z_i}\delta_{z_{i+1}} & \text{if } z_i < y \leq z_{i+1} \\ \delta_{z_N} & \text{if } y > z_N \end{cases} \tag{2.20}$$

Rowland et al. (2018) [19] showed that this projection actually minimizes the Cramér distance between the Bellman sample target distribution and the initial distribution. Bellemare et al. (2019) [3] developed S51 a variant of C51, which uses linear function approximation and updates its value distribution through the minimization of a penalised Cramér distance. The algorithm was motivated by a proof of convergence for policy evaluation.

In contrast, QDRL assumes a parametric form for the return distribution $\eta_\theta(s,a) = \frac{1}{N}\sum_{k=1}^{N}\delta_{\theta_k(s,a)}$ where the learnt parameters are now the locations of the dirac masses. The quantile projection Π_{W_1} relies on the minimisation of the 1-Wasserstein metric between the distribution of the returns and the parametrised distribution η_θ. [5] used Quantile Regression with the quantile Huber loss to stochastically find a set of minimizing parameters $\theta_1, ..., \theta_N$ and provided a proof of γ-contraction with respect to $\bar{d}_\infty$ for the application $Z \in \mathcal{Z} :\to \Pi_{W_1}\mathcal{T}_\pi Z$. The advantages of QDRL over CDRL are that we need not specify V_{MIN}, V_{MAX} or Δz, and a priori knowledge of the bounds of the distribution is not as important as in CDRL.

[6] extends the work of Dabney et al. by introducing Implicit Quantile Network (IQN) that differs from the original quantile approach in two ways. First, instead of approximating the quantile function at N fixed values $Z_{\tau_1}, ..., Z_{\tau_N}$, we approximate the quantiles with a parametrised function $Z_\tau(s,a) \simeq f(\psi(s), \phi(\tau))_a$ for some differentiable functions $f, \psi,$ and ϕ. Second, within the updates, the sampling of τ is made with a continuous distribution.

[20] proposes a unifying framework for DRL in terms of a set of statistics to be learnt, and an imputation strategy for specifying a dynamic programming update. An imputation strategy is defined as a mapping from a vector of statistics to a distribution that has those statistics. For instance, in QDRL the imputation strategy is given by $\Psi(\sigma_{1:N}) =$

$\frac{1}{N}\sum_{k=1}^{N}\delta_{\sigma_k}$. They showed that the only properties of return distributions that can be learnt exactly through Bellman updates are moments, introducing the concept of *Bellman closedness*. As this goes against the promising results of QDRL, they introduced the concept of *approximate Bellman closedness* and proved that the collection of quantiles $s_k(\eta) = F_\eta^{-1}(\frac{2k-1}{2N})$ can be approximately learnt with the appropriate imputation strategy. By analogy with the quantiles, Rowland et al. introduced a new algorithm based on Expectiles, which are to the mean of a distribution what quantiles are to the median. Formally, for $\tau \in [0,1]$, the τ-expectile of η is defined to be the minimizer $q* \in \mathbb{R}$ of the expectile regression loss $\text{ER}(q;\eta,\tau)$, equal to

$$\text{ER}(q;\eta,\tau) = \mathbb{E}_{Z\sim\eta}[[\tau \mathbb{1}_{Z>q} + (1-\tau)\mathbb{1}_{Z\leq q}](Z-q)^2] \tag{2.21}$$

Lyle et al. (2019) [12] found that the key driver in the difference in behaviour between expected and distributional alogrithms is the function approximation. With tabular representation or linear approximation, distributional methods are generally expectation-equivalent. In contrast, empirical results are in favour of distributional methods when using non-linear function approximation.

2.4 RKHS and Kernel Mean Embedding

Let $\mathcal{X}$ be a non-empty set. Kernel methods naturally arise in the ML literature when one wants to replace a scalar product $\langle x, x'\rangle$ on feature vectors $x, x' \in \mathcal{X}$ with a scalar product on transformed features $\langle \phi(x), \phi(x')\rangle_{\mathcal{F}}$. The mapping $\phi : \mathcal{X} \to \mathcal{F}$ may not be linear, $\mathcal{F}$ is usually a high-dimensional space and $\langle\cdot|\cdot\rangle_{\mathcal{F}}$ denotes the inner product of $\mathcal{F}$.

Definition 1. *[Kernel]* A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a **kernel** if there exists a Hilbert space $\mathcal{H}$ (vector space with an inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ that makes it complete for the induced metric) and a map $\phi : \mathcal{X} \to \mathcal{H}$ such that $\forall x, x' \in \mathcal{X}$

$$k(x,x') = \langle \phi(x), \phi(x')\rangle_{\mathcal{H}} \tag{2.22}$$

Notice that by the property of the inner product, a kernel has to be symmetric. $\mathcal{H}$ is called the **feature space** and ϕ is the **feature map**. It can be shown that a kernel is a positive definite application, i.e. $\forall n \geq 1, \forall (a_1,...,a_n) \in \mathbb{R}^n, \forall (x_1,...,x_n) \in \mathcal{X}^n$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j k(x_i,x_j) \geq 0 \tag{2.23}$$

Definition 2. *[RKHS]* Let $\mathcal{H}$ be a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a reproducing kernel of $\mathcal{H}$ if it satisfies the following properties

(i) $\forall x \in \mathcal{X},$ the application $k_x : y \in \mathbb{R} \to k(x, y)$ is in $\mathcal{H}$

(ii) $\forall x \in \mathcal{X}, \forall f \in \mathcal{H}, \langle f, k(\cdot, x)\rangle_{\mathcal{H}} = f(x)$ (the reproducing property)

If $\mathcal{H}$ admits a reproducing kernel, it is called a reproducing kernel Hilbert space (RKHS). Note that a reproducing kernel is clearly a kernel as $k(x, x') = \langle k((\cdot, x'), k(\cdot, x)\rangle_{\mathcal{H}}$ with the associated **canonical feature map** $\phi : x \to k(\cdot, x)$.

We have several operations that conserve the kernel structure:

(i) Given k, k_1, k_2 kernels on $\mathcal{X}$ and $\alpha > 0$, $k_1 + k_2$ and $\alpha \cdot k$ are kernels on $\mathcal{X}$.

(ii) Let $\mathcal{X}, \tilde{\mathcal{X}}$ be non-empty sets and a map $g : \mathcal{X} \to \tilde{\mathcal{X}}$. Let $\tilde{k}$ be a kernel on $\tilde{\mathcal{X}}$ then $k(x, x') \to \tilde{k}(g(x), g(x'))$ is a kernel.

(iii) Given k on $\mathcal{X}$ and l on $\mathcal{Y}$, then $\kappa((x, y), (x', y')) = k(x, x')l(y, y')$ is a kernel on $\mathcal{X} \times \mathcal{Y}$

For every symmetric positive definite kernel k on $\mathcal{X}$, the **Moore-Aronszajn** theorem [1] guarantees the existence of a unique RKHS for which k is a reproducing kernel. Kernel methods can be applied without needing to compute ϕ explicitly (Schölkopf et al. 2002 [21]). This is commonly referred to as the **kernel trick**.

Some examples of kernel include:

(i) Gaussian kernel: $k(x, x') = \exp(-\frac{\|x-x'\|_2^2}{2\sigma^2})$

(ii) Laplace kernel: $k(x, x') = \exp(-\frac{\|x-x'\|_1}{\sigma})$

(iii) Matérn kernel: $k(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)}(\frac{\sqrt{2\nu}}{\gamma}\|x - x'\|_2)^v K_\nu(\frac{\sqrt{2\nu}}{\gamma}\|x - x'\|_2)$, $\nu > 0, \gamma > 0$, where K_ν is the modified Bessel function of the second kind of order ν.

Note that these kernels are **translation invariant** as they are of the form $k(x, x') = \psi(x - x')$ on $\mathbb{R}^d$ for some ψ. Bochner's theorem (1933) provides a characterisation of complex-valued bounded continuous kernels, which are positive definite if and only if there exists a non-negative Borel measure Λ on $\mathbb{R}^d$ such that $\psi(x-x') = \int_{\mathbb{R}^d} \exp(\mathrm{i}\omega^T(x-x'))\mathrm{d}\Lambda(w)$

Kernel mean embedding extends the idea of implicit feature representation by functions in an RKHS $\mathcal{H}_k$ to the space of probability distributions by representing each probability distribution P as a mean function (Berlinet and Thomas-Agnan 2004 [4], Smola et al. 2007 [22]).

$$P \to \mu_k(P) = \mathbb{E}_{X\sim P}[k(\cdot, X)] \in \mathcal{H}_k. \tag{2.24}$$

Here the integral should be interpreted as a Bochner integral, an integral for Banach space valued functions. Using the reproducing property, if $\mathbb{E}_{X\sim P}\sqrt{k(X,X)} < \infty$, then $\mu_k(P) \in \mathcal{H}_k$ and $\forall f \in \mathcal{H}_k$, $\mathbb{E}_{X\sim P}[f(X)] = \langle f, \mu_k(P)\rangle$ (Smola et al. 2007 [22]). The moment-generating and characteristic functions can be seen as a kernel mean embedding of a distribution with respectively the kernel $k(x,x') = \exp(\langle x, x'\rangle)$ and the Fourier kernel $k(x,x') = \exp(\mathrm{i}\,\langle x, x'\rangle)$.

The Maximum Mean Discrepency (**MMD**) is the distance induced by the RKHS between two distributions P and Q

$$\begin{aligned}\mathrm{MMD}_k^2(P,Q) &= \|\mu_k(P) - \mu_k(Q)\|_{\mathcal{H}_k}^2 \\ &= \mathbb{E}_{X,X'\overset{\text{iid}}{\sim}P} k(X,X') + \mathbb{E}_{Y,Y'\overset{\text{iid}}{\sim}Q} k(Y,Y') - 2\mathbb{E}_{X,Y\sim P,Q} k(X,Y)\end{aligned} \tag{2.25}$$

The MMD can also be written as

$$\mathrm{MMD}_k(P,Q) = \sup_{f\in\mathcal{H}_k:\|f\|_{\mathcal{H}_k}\leq 1} |\mathbb{E}_{X\sim P} f(X) - \mathbb{E}_{X\sim Q} f(X)| \tag{2.26}$$

The function f^* where the supremum of (2.26) is attained, is shown to be proportional to the difference $\mu_k(P) - \mu_k(Q)$ and is called the **witness function**. A kernel k is said to be characteristic if the map $\mu : P \to \mu_k(P)$ is injective. Gaussian, Matérn and Laplacian kernels belong to this family of kernels. For characteristic kernels, the MMD defines a proper metric on proability distributions since $\mathrm{MMD}_k(P,Q) = 0$ implies $P = Q$ a.s. .

Let (X,Y) be a random variable taking values on $X \times Y$ and $\mathcal{H}_k$ and $\mathcal{G}_l$ be RKHS with measurable kernels on $\mathcal{X}$, and $\mathcal{Y}$, respectively. We assume that $\mathcal{H}_k \subset L^2(P_X)$ and $\mathcal{G}_l \subset L^2(P_Y)$. The (uncentered) **cross-covariance** operator $\tilde{C}_{YX} : \mathcal{H} \to \mathcal{G}$ can be defined in terms of the tensor product $\varphi(Y) \otimes \phi(X)$ in a tensor product feature space $\mathcal{G} \otimes \mathcal{H}$ as

$$\tilde{C}_{YX} = \mathbb{E}_{YX}[\varphi(Y) \otimes \phi(X)] = \mu(P_{YX}) \tag{2.27}$$

where P_{YX} denotes the joint distribution of (X,Y) and ϕ (resp. φ) are the canonical feature map corresponding to k (resp. l). The centered cross-covarariance operator is $C_{YX} = \mu(P_{YX}) - \mu(P_Y) \otimes \mu(P_X)$. The cross-covariance operator C_{YX} is a generalisation of the covariance operator as $\forall g \in \mathcal{G}_l, \forall f \in \mathcal{H}_k, \langle g, C_{YX} f\rangle_{\mathcal{G}_l} = \mathrm{Cov}[g(Y), f(X)]$. If $X = Y$, we call C_{XX} the **covariance operator**.

Let $\mu_{Y|X} : \mathcal{H} \to \mathcal{G}$ and $\mu_{Y|x} \in \mathcal{G}$ be the conditional mean embeddings of the conditional

distribution $P(Y|X)$ and $P(Y|X=x)$ such that they satisfy

$$\mu_{Y|x} = \mathbb{E}_{Y|x}[\varphi(Y)|X=x] = \mu_{Y|X}k(x,\cdot) \tag{2.28}$$

$$\mathbb{E}_{Y|x}[g(Y)|X=x] = \langle g, \mu_{Y|x}\rangle_{G_l} \forall g \in \mathcal{G}_l \tag{2.29}$$

Song et al. (2009 [24], 2013 [23]) gave a proper definition for the conditional mean embedding $\mathcal{U}_{Y|X}$ and $\mathcal{U}_{Y|x}$

$$\mu_{Y|X} = C_{YX}C_{XX}^{-1} \tag{2.30}$$

$$\mu_{Y|x} = C_{YX}C_{XX}^{-1}k(x,\cdot) \tag{2.31}$$

This definition holds for the properties (2.28) and (2.29) because of a Theorem by Fukumizu et al. (2004) [9] that states that if $\mathbb{E}_{YX}[g(Y)|X=\cdot] \in \mathcal{H}$ for $g \in \mathcal{G}$, then $C_{XX}\mathbb{E}_{YX}[g(Y)|X=\cdot] = C_{XY}g$. Note that the conditional mean operator might not exist in the continuous domain but this issue may be circumvented by including regularisation $C_{YX}(C_{XX}+\lambda\mathbf{I})^{-1}k(x,\cdot)$, λ being the regularisation parameter. In practice, to estimate the conditional mean embedding we have access to n i.i.d. samples $(x_1,y_1),...,(x_n,y_n)$. If we denote $\mathbf{\Phi} = [\varphi(y_1),...,\varphi(y_n)]^T$, $\mathbf{\Upsilon} = [\phi(x_1),...,\phi(x_n)]^T$ and their associated Gram matrix $\mathbf{K} = \mathbf{\Upsilon}^T\mathbf{\Upsilon}$ and $\mathbf{L} = \mathbf{\Phi}^T\mathbf{\Phi}$, then the conditonal mean embeddding $\hat{\mu}_{Y|x}$ can be estimated by

$$\mathbf{\Phi}(\mathbf{K}+n\lambda\mathbf{I}_n)^{-1}\mathbf{k}_x \tag{2.32}$$

where $\mathbf{k}_x = \Upsilon^T k(x,\cdot)$. This can be written as a scalar product:

$$\hat{\mu}_{Y|x} = \sum_{i=1}^{n}\varphi(y_i)\beta_i(x) = \mathbf{\Phi}\boldsymbol{\beta}(x) \tag{2.33}$$

where $\boldsymbol{\beta}(x) = [\beta_1(x),...,\beta_n(x)] = (\mathbf{K}+n\lambda\mathbf{I}_n)^{-1}\mathbf{k}_x$.

Given a set of samples $(x_1,y_1),...,(x_n,y_n) \in \mathcal{X}\times\mathcal{G}$, Grünewälder et al. (2012) [7] shows that $\hat{\mu}_{Y|X}$ can in fact be obtained by minimizing the regularised loss of an underlying regression problem: $\mathcal{E}_\lambda(f) = \sum_{i=1}^{n}\|z_i - f(x_i)\|_{\mathcal{G}}^2 + \lambda\|f\|_{\mathcal{H}_\mathcal{G}}^2$ where $\mathcal{G}$ is a Hilbert space and $\mathcal{H}_\mathcal{G}$ is an RKHS of vector-valued functions from $\mathcal{X}$ to $\mathcal{G}$.

Chapter 3

General Value Distribution

In this chapter, we will combine the distributional approach of Reinforcement Learning with the General Value Function framework. We coin this new framework, General Value Distribution as it extends Distributional RL with a GVF parametrisation.

3.1 GVD framework

Essentially, a value function is the expectation of a conditional return. The "General" aspect of it simply means we can answer a larger set of questions through a γ function which depends on the states. There is a priori no contradiction of using a General Value Function parametrisation with the algorithms used for distributional RL. The main difference between GVF and DRL is their theoretical point of views. GVF Bellman equation is based on the expected sum of cumulants whereas DRL Bellman equation is expressed in terms of the returns distribution. Just as GVF extended standard value function by introducing a termination signal function γ, we can introduce the same termination function into Distributional RL to form the GVD framework. Therefore, GVD differs to DRL simply by adding the γ function to the set of question functions. As for the answer functions, they depend on the distributional algorithm used to solve the task. For instance, one could mention the number of quantiles for the QDRL algorithm or the maximum and minimum return values for CDRL. More generally, if we use a neural network to output parameters of a distribution, the architecture of this network is also an answer function.

Formally, a GVD estimates for each state-action pair the distribution $\eta_\pi(s,a)$ of the r.v. $\tilde{Z}_\pi(s,a) = \tilde{G}_t|(S_t = s, A_t = a, A_{t+1:T} \sim \pi)$, where the return $\tilde{G}_t$ is defined as the following sum $\sum_{k=t}^{T} \prod_{j=t+1}^{k} \gamma(S_j)c(S_{k+1})$.

As for GVF, the three inputs for a GVD are simply:

- the target policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$
- the termination signal $\gamma : \mathcal{S} \to [0, 1]$
- the cumulant function $c : \mathcal{S} \to \mathbb{R}$, which can be stochastic.

This induces a slight change for the distributional Bellman equations and thus on the update rules of the algorithms.

$$\begin{aligned} \tilde{Z}_\pi(s,a) &\overset{D}{=} \mathcal{T}^\pi \tilde{Z}_\pi(s,a) \\ &\overset{D}{=} C(s,a) + \gamma(S')\tilde{Z}_\pi(S',A') \end{aligned} \tag{3.1}$$

with $(S', R) \sim P(\cdot|s, a)$, $A' \sim \pi(\cdot|S')$. For convenience, we will simply refer to $\tilde{Z}$ by Z and drop the tilde sign. We can also express this equation in terms of the distributions $\mathcal{T}^\pi \eta_\pi$ and η_π, assuming that they exist [19]. This will be useful since DRL algorithms assume a parametric form for these distributions. If $\forall s, \gamma(s) > 0$, we obtain the distribution of the RHS of Equation (3.1) by integrating over all possible cumulants, current states and next actions, we get $\forall z \in \mathbb{R}$

$$\begin{aligned} \mathcal{T}^\pi \eta_\pi^{(s,a)}(z) &= \int_{-\infty}^{\infty} \sum_{s',a'} p(c, s'|s, a)\pi(a'|s')\eta_\pi^{(s',a')}((z-c)/\gamma(s'))dc \\ &= \mathbb{E}_{c,s' \sim p(\cdot|s,a)} \mathbb{E}_{a' \sim \pi(a'|s')} (f_{c,\gamma(s')})_\# \eta_\pi^{(s',a')}(z) \end{aligned} \tag{3.2}$$

Where we noted $f_{c,\gamma}(z) = c + \gamma z$ and simplified notation with the pushforward measure.

Definition 3. *For a measurable function $g : \mathbb{R} \to \mathbb{R}$, and a measure $\nu \in \mathcal{P}(\mathbb{R})$, we note $g_\#\nu \in \mathcal{P}(\mathbb{R})$ the **pushforward** measure of ν by g which is defined as follows : for every measurable set $A \subset \mathbb{R}$, $g_\#\nu(A) = \nu(g^{-1}(A))$.*

Indeed, for two independent random variables U, V the density of $W = U + V$ is

$$f_{U+V}(w) = \int_{\mathbb{R}} f_{U,V}(u, w-u)du = \int_{\mathbb{R}} f_U(u) f_V(w-u|U=u)du \tag{3.3}$$

So equation (3.2) holds because $C(s, a)$ and $Z_\pi(s', a')$ are assumed to be independent. Similarly, the distributional optimality Bellman equation is obtained when π is the greedy policy with respect to the q values.

Ideally, we would like GVD to perform as good as DRL algorithms, but with a GVF perspective. That is, we would like

- to assign multiple GVDs to a Horde, that can be learnt in parallel and off-policy

from a single stream of experience.

- to have GVDs that can do off-policy evaluation and off-policy control.

Distributional algorithms, as described in the original papers, were mainly used for a Q-learning task and therefore estimate directly the optimal state-action value distribution. Though this is what is of interest in most Reinforcement Learning settings, we wish to demonstrate capabilities for state-action evaluation as it is an essential focus of the GVD framework. We will work with GVDs that build on QDRL as it is a well-studied, flexible and effective algorithm.

3.2 Classical GVF algorithms

Before moving on to presenting GVD techniques, we provide further details on evaluation algorithms mentioned in Section 2.2, namely GTD(λ) and GQ(λ). We also present a control algorithm, called greedy-GQ, introduced by Maei. et al in 2010 [15], which naturally extends GQ(λ) to do policy iteration.

Algorithm 1 GTD(λ)

Inputs: The policy π to be evaluated, the behaviour policy b, a feature function $x : \mathcal{S} \to \mathbb{R}^d$, a termination function $\gamma : \mathcal{S} \to [0, 1]$
Parameters: Step size $\alpha > 0$, Step size $\alpha_h > 0$, trace decay rate $\lambda \in [0, 1]$
Output: $v(s)$ for all $s \in \mathcal{S}$
Initialise: $\boldsymbol{w} \leftarrow 0$; $\boldsymbol{h} \leftarrow 0$
for *episode in episodes* **do**
  $\boldsymbol{e} \leftarrow 0$; $S \leftarrow S_0$; $\boldsymbol{x} \leftarrow x(S)$; $\gamma_{\text{last}} \leftarrow \gamma_0$
  for *step in episode* **do**
    Select A according to b
    Take action A
    Observe C, S', $\boldsymbol{x'}$ and $\gamma_{\text{next}} = \gamma(S')$
    $\rho \leftarrow \frac{\pi(A|S)}{b(A|S)}$
    $\delta \leftarrow C + \gamma_{\text{next}} \boldsymbol{x'}^T \boldsymbol{w} - \boldsymbol{x}^T \boldsymbol{w}$
    $\boldsymbol{e} \leftarrow \rho(\gamma_{\text{last}} \lambda \boldsymbol{e} + \boldsymbol{x})$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(\delta \boldsymbol{e} - \gamma_{\text{next}}(1 - \lambda)(\boldsymbol{e}^T \boldsymbol{h})\boldsymbol{x'})$
    $\boldsymbol{h} \leftarrow \boldsymbol{h} + \alpha_h(\delta \boldsymbol{e} - (\boldsymbol{h}^T \boldsymbol{x})\boldsymbol{x})$
    $S \leftarrow S'$; $x \leftarrow x'$; $\gamma_{\text{last}} \leftarrow \gamma_{\text{next}}$
  end
end

The idea behind greedy-GQ is to alternate between a "latent" policy update – improving the policy with respect to the current state-action values – and the learning of state-action values for this updated policy. The policy improvement can be greedy, ϵ-greedy, soft-max,

etc. with respect to the q values. The policy learning is latent as the behaviour policy remains unchanged. Showing that this method converges steadily to the desired policy is not the focus of this dissertation and interested readers can refer to [14] for further details. However, we can at least point out the intuitions behind this method. First, let us note π_0 the current policy of our GVF. Updating the policy in a greedy fashion results in a new policy $\pi_1 : \forall(s,a) \in \mathcal{A} \times \mathcal{S}, \pi_1(a|s) = \arg\max_a q_{\pi_0}(s,a)$. Since $\forall s \in \mathcal{S}, v_{\pi_0}(s) = \mathbb{E}_{a\sim\pi_0(\cdot|s)} q_{\pi_0}(s,a)$, we immediately have that $\forall s \in \mathcal{S}, v_{\pi_0}(s) \leq q_{\pi_0}(s, \pi_1(s))$. Therefore,

$$
\begin{aligned}
v_{\pi_0}(s) &\leq q_{\pi_0}(s, \pi_1(s)) && (3.4)\\
&= \mathbb{E}_{r_1,s_1\sim p(\cdot|s,\pi_1(s))}[r_1 + \gamma(s_1) v_{\pi_0}(s_1)] && (3.5)\\
&\leq \mathbb{E}_{r_1,s_1\sim p(\cdot|s,\pi_1(s))}[r_1 + \gamma(s_1) q_{\pi_0}(s_1, \pi_1(s_1))] && (3.6)\\
&= \mathbb{E}_{r_1,s_1,r_2,s_2}[r_1 + \gamma(s_1) r_2 + \gamma(s_1)\gamma(s_2) v_{\pi_0}(s_2)] && (3.7)\\
&\leq \mathbb{E}_{r_1,s_1,r_2,s_2}[r_1 + \gamma(s_1) r_2 + \gamma(s_1)\gamma(s_2) q_{\pi_0}(s_2, \pi_1(s_2))] && (3.8)\\
&... && (3.9)\\
&\leq v_{\pi_1}(s) && (3.10)
\end{aligned}
$$

We obtain the last inequality by induction and we write the value function of the RHS with subscript π_1 since all actions until reaching the terminal state are selected according to π_1. Thus, greedifying the policy according to the q values results in a new policy that is not worse than the previous one. This result is an example of the policy improvement theorem [25]. This new policy affects future importance sampling ratio $\rho(s,a) = \frac{\pi(a|s)}{b(a|s)}$ used in the GQ updates, which enables a correction for taking the immediate action a in state s under policy b instead of policy π. It also affects the next mean state-action feature $\bar{\boldsymbol{x}}(s') = \mathbb{E}_{a'\sim\pi_1(\cdot|s')} \boldsymbol{x}(s', a')$ which is used to form a target derived from Expected SARSA, regardless of b.

Algorithm 2 GQ(λ)

Inputs: The policy π to be evaluated, the behaviour policy b, a feature function $x : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{d \times |\mathcal{A}|}$, a termination function $\gamma : \mathcal{S} \to [0, 1]$

Parameters: Step size $\alpha > 0$, Step size $\alpha_h > 0$, trace decay rate $\lambda \in [0, 1]$

Output: $q(s, a)$ for all $s \in \mathcal{S}$

Initialise: $\boldsymbol{w} \leftarrow 0$; $\boldsymbol{h} \leftarrow 0$

for *episode in episodes* **do**

  $\boldsymbol{e} \leftarrow 0$; $S \leftarrow S_0$; $\gamma_{\text{last}} \leftarrow \gamma_0$

  for *step in episode* **do**

    Select A according to b

    Take action A

    Observe C, S', and $\gamma_{\text{current}} = \gamma(S')$

    $\rho \leftarrow \frac{\pi(A|S)}{b(A|S)}$

    $\bar{\boldsymbol{x}} \leftarrow 0$; $\boldsymbol{x} \leftarrow \boldsymbol{x}(S, A)$

    for $a' \in \mathcal{A}(S')$ **do**

      $\bar{\boldsymbol{x}} \leftarrow \bar{\boldsymbol{x}} + \pi(a'|S')\boldsymbol{x}(S', a')$

    end

    $\delta \leftarrow C + \gamma_{\text{current}}\bar{\boldsymbol{x}}^T\boldsymbol{w} - \boldsymbol{x}^T\boldsymbol{w}$

    $\boldsymbol{e} \leftarrow \rho\gamma_{\text{last}}\lambda\boldsymbol{e} + \boldsymbol{x}(S, A)$

    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha(\delta\boldsymbol{e} - \gamma_{\text{current}}(1 - \lambda)(\boldsymbol{e}^T\boldsymbol{h})\bar{\boldsymbol{x}})$

    $\boldsymbol{h} \leftarrow \boldsymbol{h} + \alpha_h(\delta\boldsymbol{e} - (\boldsymbol{h}^T\boldsymbol{x})\boldsymbol{x})$

    $S \leftarrow S'$; $\gamma_{\text{last}} \leftarrow \gamma_{\text{current}}$

  end

end

Algorithm 3 greedy-GQ(λ)

Inputs: An initial policy π, the behaviour policy b, a feature function $x : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{d \times |\mathcal{A}|}$, a termination function $\gamma : \mathcal{S} \to [0, 1]$, a policy improvement function $\Pi : w \to \pi$

Parameters: Step size $\alpha > 0$, Step size $\alpha_h > 0$, trade decay rate $\lambda \in [0, 1]$

Output: An improved policy π^*; $q_{\pi^*}(s, a)$ for all $s, a \in \mathcal{S} \times \mathcal{A}$

Initialise: $\boldsymbol{w} \leftarrow 0$; $\boldsymbol{h} \leftarrow 0$

for *episode in episodes* **do**

  $\boldsymbol{e} \leftarrow 0$; $S \leftarrow S_0$; $\gamma_{\text{last}} \leftarrow \gamma_0$

  for *step in episode* **do**

    Take GQ(λ) updates

    $\pi \leftarrow \Pi(w)$

  end

end

3.3 GVD algorithms with QDRL

We now recall QDRL setting, with the addition of the termination signal γ. We call this setting **QGVD**. Let N be the number of quantiles. Let $\tau_0 = 0$ and $\tau_i = \frac{i}{N}$ for $i \in 1, ..., N$. We assume a parametric distribution of the form $\eta_\theta^{(s,a)} = \frac{1}{N}\sum_{i=1}^{N} \delta_{\theta_i(s,a)}$. We thus have by construction that $\mathbb{P}(Z_\theta \leq \theta_i(s,a)) = \tau_i$ and that for $t \in (\tau_{i-1}, \tau_i]$, $F_\theta^{-1}(t) = \theta_i(s,a)$.

Let $\mathcal{Z}_\mathcal{Q}$ be the space of random variables with uniform distribution on N diracs. For a distribution η with cdf F, and a distribution $\eta_\theta \in \mathcal{Z}_\mathcal{Q}$ with cdf F_θ, we wish to find the minimising θ-values of the 1-Wasserstein metric

$$\begin{aligned} d_1(\eta, \eta_\theta) &= \int_0^1 |F^{-1}(\omega) - F_\theta^{-1}(\omega)| d\omega \\ &= \sum_{i=1}^{N} \int_{\tau_{i-1}}^{\tau i} |F^{-1}(\omega) - \theta_i| d\omega \end{aligned} \tag{3.11}$$

Lemma 2 from the original paper [5] claims that for any $\tau, \tau' \in [0,1]$, the set of minimising values of the integral $\int_\tau^{\tau'} |F^{-1}(\omega) - \theta| d\omega$ is $\{\theta, F(\theta) = \frac{\tau+\tau'}{2}\}$. Thus, $F^{-1}(\frac{\tau+\tau'}{2})$ is a valid minimiser and is the unique minimiser if F is continuous at $\frac{\tau+\tau'}{2}$. Since $d_1(\eta, \eta_\theta)$ is a sum of positive integrals, the minimising θ-values are $(F^{-1}(\hat{\tau}_1), ..., F^{-1}(\hat{\tau}_N))$ with $\hat{\tau}_i = \frac{\tau_{i-1}+\tau_i}{2}$ and we define the projection $\Pi_{W_1} : \eta \to \frac{1}{N}\sum_{i=1}^{N} \delta_{F_\eta^{-1}(\hat{\tau}_i)}$.

$\Pi_{W_1}\mathcal{T}^\pi\eta_\pi$ cannot be directly computed in practice. However, for $\tau \in [0,1]$, the value $F^{-1}(\tau)$ can be characterized as the minimiser over $\theta \in \mathbb{R}$ of the Quantile Regression loss $\mathcal{L}_{QR}(\theta, \eta, \tau) = \mathbb{E}_{Z\sim\eta}[[\tau \mathbb{1}_{Z>\theta} + (1-\tau)\mathbb{1}_{Z\leq\theta}]|Z - \theta|]$. Therefore, we can approximately minimise the 1-Wasserstein distance of the target distribution by taking a stochastic gradient step. Since,

$$\begin{aligned} \mathcal{L}_{QR}(\theta, \mathcal{T}^\pi\eta^{(s,a)}, \tau) &= \mathbb{E}_{Z\sim\mathcal{T}^\pi\eta^{(s,a)}}[[\tau \mathbb{1}_{Z>\theta} + (1-\tau)\mathbb{1}_{Z\leq\theta}]|Z - \theta|] \\ &\overset{(a)}{=} \mathbb{E}_{c,s'\sim P(\cdot|s,a)}\mathbb{E}_{a'\sim\pi(\cdot|s')}\mathcal{L}_{QR}(\theta, (f_{c,\gamma(s')})_\#\eta_\pi^{(s',a')}, \tau) \\ &= \mathbb{E}_{c,s'\sim P(\cdot|s,a)}\mathbb{E}_{a'\sim\pi(\cdot|s')}\mathcal{L}_{QR}(\theta, \frac{1}{N}\sum_{i=1}^{N}\delta_{\phi_i^c(s',a')}, \tau) \\ &= \mathbb{E}_{c,s'\sim P(\cdot|s,a)}\mathbb{E}_{a'\sim\pi(\cdot|s')}\mathcal{L}_{QR}(\theta, \eta_{\phi^c}^{(s',a')}, \tau) \end{aligned} \tag{3.12}$$

where $\phi_i^c(s',a') = c + \gamma(s')\theta_i(s',a')$. We obtain (a) with Fubini to exchange integral orders.

Therefore,

$$\begin{aligned}\nabla_\theta \mathcal{L}_{QR}(\theta, \mathcal{T}^\pi \eta^{(s,a)}, \tau) &= \nabla_\theta \mathbb{E}_{c,s' \sim P(\cdot|s,a)} \mathbb{E}_{a' \sim \pi(\cdot|s')} \mathcal{L}_{QR}(\theta, \eta_{\phi^c}^{(s',a')}, \tau) \\ &\overset{(b)}{=} \mathbb{E}_{c,s' \sim P(\cdot|s,a)} \mathbb{E}_{a' \sim \pi(\cdot|s')} \nabla_\theta \mathcal{L}_{QR}(\theta, \eta_{\phi^c}^{(s',a')}, \tau)\end{aligned} \tag{3.13}$$

where in (b) we assumed that we could differentiate under the integral. This means that if we sample $(s, a, c, \gamma, s', a')$ through a policy π, we can compute an unbiased estimate of the gradient $\nabla_\theta \mathcal{L}_{QR}(\theta, \mathcal{T}^\pi \eta^{(s,a)}, \tau)$ with $\nabla_\theta \mathcal{L}_{QR}(\theta, (f_{c,\gamma(s')})_\# \eta_\pi^{(s',a')}, \tau)$ [20]. Furthermore, we actually do not need to store samples of a' sampled from π when we generate transitions. We can instead wander around the state space with the behaviour policy b and sample a' from π for each (s, a, c, γ, s'). It is therefore a natural off-policy evaluation method, which has been described for CDRL in [2].

We can also use $\nabla_\theta \mathbb{E}_{a' \sim \pi(\cdot|s')} \mathcal{L}_{QR}(\theta, (f_{c,\gamma(s')})_\# \eta_\pi^{(s',a')}, \tau)$ to form an unbiased estimate, which is the gradient of the loss averaged over all actions. We will use this as the basis of our new off-policy evaluation algorithm. Note that QDRL uses the quantile Huber Loss which is a smoother variant than the QR loss and which is defined as

$$r_\tau^\kappa(u) = |\tau - \mathbb{1}_{u<0}| [\frac{1}{2} u^2 \mathbb{1}_{|u| \leq \kappa} + \kappa(|u| - \frac{1}{2}\kappa) \mathbb{1}_{|u|>\kappa}] \tag{3.14}$$

For $\kappa = 0$, we get back the standard QR loss.

Algorithm 4 Q-Learning - QGVD (adapted from QR-DQN)

Inputs: An initial policy π, the behaviour policy b, a feature function $x : \mathcal{S} \to \mathbb{R}^d$, a termination function $\gamma : \mathcal{S} \to [0, 1]$, a Neural Network $\Theta : \boldsymbol{x} \in \mathbb{R}^d \to \boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{A}| \times N}$

Parameters: Optimizer with step size $\eta > 0$, the number of quantiles N, quantile Huber loss parameter $\kappa \in [0, 1]$, a risk parameter β

Output: Updated Θ

Initialise: $\Theta \leftarrow \Theta_0$; $\tau = (\frac{1}{2N}, \frac{3}{2N}, ..., \frac{2N-1}{2N})$

for *episode in episodes* **do**

  $S \leftarrow S_0$

  for *step in episode* **do**

    Select A according to b

    Take action A

    Observe C, S', and $\gamma_{\text{current}} = \gamma(S')$

    $a^* = \arg\max_{a'} q_\pi(S', a') = \frac{1}{N} \sum_{j=1}^{N} \boldsymbol{\theta}_j(S', a')$

    $\forall j \in \{1, ..., N\}, \mathcal{T}\boldsymbol{\theta}_j \leftarrow C + \gamma_{\text{current}} \boldsymbol{\theta}_j(S', a^*)$

    $\mathcal{L} = \sum_{i=1}^{N} \mathbb{E}_j r_{\tau_i}^{\kappa}(\mathcal{T}\boldsymbol{\theta}_j - \boldsymbol{\theta}_i(s, a))$, where $r_{\tau_i}^{\kappa}$ is the quantile huber loss.

    Take an optimizer step to minimize $\mathcal{L}$

    $S \leftarrow S'$

  end

end

We present in Algorithm 4 the **QR-DQN** method from Dabney et al. [5] with a GVD parametrisation. It is a control method with QDRL, that builds on a Neural Network representation.

We can adapt this algorithm to do policy evaluation by replacing the selection of the action a^* with a sample a' from the target policy π. In the case of QDRL, this technique is not mentioned in the original paper, but is now new as it was exposed in [2]. For the sake of distinction, we call this evaluation method **One Sample - QGVD**. However, sampling only one action induces another level of stochasticity, and we suggest a new algorithm for policy evaluation based on the loss averaging over all actions. This does not change the theoretical guarantees of the method for only the stochastic approximation is changed.

This gives us the following algorithm, which we call **Average Loss - QGVD**.

Algorithm 5 Average Loss - QGVD

Inputs: The policy π to be evaluated, the behaviour policy b, a feature function $x : \mathcal{S} \to \mathbb{R}^d$, a termination function $\gamma : \mathcal{S} \to [0, 1]$, a Neural Network $\Theta : \boldsymbol{x} \in \mathbb{R}^d \to \boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{A}| \times N}$

Parameters: Optimizer with step size $\eta > 0$, the number of quantiles N, quantile Huber loss parameter $\kappa \in [0, 1]$

Output: Updated Θ

Initialise: $\Theta \leftarrow \Theta_0$; $\tau = (\frac{1}{2N}, \frac{3}{2N}, ..., \frac{2N-1}{2N})$

for *episode in episodes* **do**
  $S \leftarrow S_0$
  for *step in episode* **do**
    Select A according to b
    Take action A
    Observe C, S', and $\gamma_{\text{current}} = \gamma(S')$
    $\mathcal{L} = 0$; $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}(S, A)$
    for $a' \in \mathcal{A}(S')$ **do**
      $\forall j \in \{1, ..., N\}$, $\mathcal{T}\boldsymbol{\theta}_j \leftarrow C + \gamma_{\text{current}}\boldsymbol{\theta}_j(S', a')$
      $\mathcal{L} = \mathcal{L} + \pi(a'|S') \sum_{i=1}^{N} \mathbb{E}_j r^{\kappa}_{\tau_i}(\mathcal{T}\boldsymbol{\theta}_j - \boldsymbol{\theta}_i(s, a))$, where $r^{\kappa}_{\tau_i}$ is the quantile huber loss.
    end
    Take an optimizer step to minimize $\mathcal{L}$
    $S \leftarrow S'$
  end
end

Besides, inspired by IQN and Particle Value Functions [13], we suggest a new algorithm that builds on utility theory to take profit of the distribution over the returns to define control policies. Indeed, in QR-DQN, we select the action that maximises the expectation of the returns, just as in expected Reinforcement Learning. We rapidly recall some results from [13].

A utility function $u : \mathbb{R} \to \mathbb{R}$ is an invertible non-decreasing function that specifies a ranking over possible returns $\tilde{G}_t$. We focus on exponential utilities $u_{\text{exp}}(x) = \text{sign}(\beta) \exp(\beta)$ where $\beta \in \mathbb{R}$ and we define the associated q_β value as follows:

$$\begin{aligned} q^{\beta}_{\pi}(s, a) &= u^{-1}_{\text{exp}}(\mathbb{E}[u_{\text{exp}}(\tilde{G}_t)|S_t = s, A_t = a, A_{t+1:T} \sim \pi] \\ &= \frac{1}{\beta} \log \mathbb{E}[\exp(\beta \tilde{G}_t)|S_t = s, A_t = a, A_{t+1:T} \sim \pi] \end{aligned} \tag{3.15}$$

It can be shown that for $\beta \to 0$ we recover the usual q value. Besides, Maddison et al. [13] claim that $v^{\beta}_{\pi}(s)$, the corresponding state value to $q^{\beta}_{\pi}(s, a)$, is risk-seeking for $\beta > 0$ and risk-averse for $\beta < 0$. The proof of this result is given for an undiscounted sum of rewards, but holds for our general return. It is based on the fact that $\beta \to v^{\beta}_{\pi}(s)$ is a

non-decreasing function for every policy π. We can obtain the same property for $q_\pi^\beta(s,a)$, the only change is that we condition by the last action a.

Lemma 1. *The function $\beta \to q_\pi^\beta(s,a)$ is a non-decreasing function for every policy π, state-action pair $s, a \in \mathcal{S} \times \mathcal{A}$.*

Proof. Let $s, a \in \mathcal{S} \times \mathcal{A}$ and π be a policy. Let $\beta, \alpha \neq 0$ such that $\beta \geq \alpha$. Then, we either have $\frac{\beta}{\alpha} \geq 1$ of $\frac{\beta}{\alpha} < 0$. Using the fact that $x \to x^p$ is convex on $\mathbb{R}^+$ for $p < 0$ and $p \geq 1$, we have

$$\begin{aligned} \mathbb{E}_\pi[\exp(\beta\tilde{G}_t)|S_t = s, A_t = a]) &= \mathbb{E}_\pi[\exp(\alpha\tilde{G}_t)^{\frac{\beta}{\alpha}}|S_t = s, A_t = a]) \\ &\geq \mathbb{E}_\pi[\exp(\alpha\tilde{G}_t)|S_t = s, A_t = a]))^{\frac{\beta}{\alpha}} \end{aligned} \tag{3.16}$$

using Jensen's inequality. Taking log of both sides yields the result for $\beta > 0$. If $\beta < 0$, then $\alpha < 0$ and we get

$$\begin{aligned} \mathbb{E}_\pi[\exp(\beta\tilde{G}_t)|S_t = s, A_t = a] &= \mathbb{E}_\pi[\exp(-|\beta|\tilde{G}_t)|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[\exp(\alpha\tilde{G}_t)^{\frac{|\beta|}{|\alpha|}}|S_t = s, A_t = a] \\ &\geq \mathbb{E}_\pi[\exp(\alpha\tilde{G}_t)|S_t = s, A_t = a]^{\frac{|\beta|}{|\alpha|}} \end{aligned} \tag{3.17}$$

We obtain the result in a similar way and by multiplying twice by -1 to recover α and β. □

Thanks to Lemma 1, we know that q_π^β is also risk sensitive. We therefore design a variant of Q-Learning that relies on q_π^β and that we call **risk-sensitive Q-Learning - QGVD**. Our method should induce differences in policy iteration when confronted with return distributions of different risk levels.

Algorithm 6 risk-sensitive Q-Learning - QGVD

Inputs: An initial policy π, the behaviour policy b, a feature function $x : \mathcal{S} \to \mathbb{R}^d$, a termination function $\gamma : \mathcal{S} \to [0, 1]$, a Neural Network $\Theta : \boldsymbol{x} \in \mathbb{R}^d \to \boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{A}| \times N}$

Parameters: Optimizer with step size $\eta > 0$, the number of quantiles N, quantile Huber loss parameter $\kappa \in [0, 1]$, a risk parameter β

Output: Updated Θ

Initialise: $\Theta \leftarrow \Theta_0$; $\tau = (\frac{1}{2N}, \frac{3}{2N}, ..., \frac{2N-1}{2N})$

for *episode in episodes* **do**
  $S \leftarrow S_0$
  for *step in episode* **do**
    Select A according to b
    Take action A
    Observe C, S', and $\gamma_{\text{current}} = \gamma(S')$
    if $\beta = 0$ **then**
      $a^* = \arg\max_{a'} q_\pi(S', a') = \frac{1}{N} \sum_{j=1}^{N} \boldsymbol{\theta}_j(S', a')$
    else
      $a^* = \arg\max_{a'} q_\pi^\beta(S', a') = \frac{1}{\beta} \log[\frac{1}{N} \sum_{j=1}^{N} \exp(\beta \boldsymbol{\theta}_j(S', a'))]$
    $\forall j \in \{1, ..., N\}, \mathcal{T}\boldsymbol{\theta}_j \leftarrow C + \gamma_{\text{current}} \boldsymbol{\theta}_j(S', a^*)$
    $\mathcal{L} = \sum_{i=1}^{N} \mathbb{E}_j r_{\tau_i}^\kappa(\mathcal{T}\boldsymbol{\theta}_j - \boldsymbol{\theta}_i(s, a))$, where $r_{\tau_i}^\kappa$ is the quantile huber loss.
    Take an optimizer step to minimize $\mathcal{L}$
    $S \leftarrow S'$
  end
end

3.4 Theoretical guarantees for GVD

In this section, we will mainly adapt the work from Bellemare et al. [2], Dabney et al. [5] and Rowland et al. [20] to present theoretical guarantees for GVD and especially QGVD. Concretely, adapting the proofs comes down to using the termination signal function instead of a constant γ rate.

Lemma 2. *Let $\gamma : S \to [0, 1)$ and $\gamma_{sup} = \sup_{s \in \mathcal{S}} \gamma(s)$, then the application $\mathcal{T}^\pi : \mathcal{P}(\mathbb{R}) \to \mathcal{P}(\mathbb{R})$ is a γ_{sup}-contraction in $\bar{d}_p$ if $\gamma_{sup} < 1$ and is a non-expansion if $\gamma_{sup} = 1$.*

Proof. Lemma 3 from Dabney et al. assures us that for a transition to any state s', with termination signal $\gamma(s')$, $\mathcal{T}_{\gamma(s')}^\pi$ is a $\gamma(s')$-contraction in $\bar{d}_p$ if $\gamma(s') < 1$. Thus, for $\eta_1, \eta_2 \in \mathcal{P}(\mathbb{R})$, we have $\forall s' \in \mathcal{S}$, $\bar{d}_p(\mathcal{T}^\pi \eta_1, \mathcal{T}^\pi \eta_2) < \gamma(s') \bar{d}_p(\eta_1, \eta_2) < \gamma_{\text{sup}} \bar{d}_p(\eta_1, \eta_2)$. Hence, the result. □

Lemma 2 and Banach's fixed point theorem guarantee that the successive application of $\mathcal{T}^\pi$ to a distribution η converges to a unique fixed point and that this fixed point is the true

distribution η_π [2].

However, using the distributional Bellman operator directly is impractical as distributions live in an infinite-dimensional space. Hence, the main techniques resort to project the distribution onto a finite-dimensional space by minimising a metric. For quantile regression, this projection is the application $\Pi_{W_1} : \eta \in \mathcal{P}(\mathbb{R}) \to \frac{1}{N}\sum_{i=1}^{N} \delta_{F_\eta^{-1}(\hat{\tau}_i)}$.

Lemma 3. *The application $\Pi_{W_1}\mathcal{T}^\pi : \mathcal{P}(\mathbb{R}) \to \mathcal{P}(\mathbb{R})$ is a γ_{sup}-contraction in $\bar{d}_\infty$ if $\gamma_{sup} < 1$ and is a non-expansion if $\gamma_{sup} = 1$.*

Proof. Similarly, we use Proposition 2 from Dabney et al. [5] which states the result for a constant γ. We immediately have that for $\eta_1, \eta_2 \in \mathcal{P}(\mathbb{R}), \forall s' \in \mathcal{S}, \bar{d}_\infty(\Pi_{W_1}\mathcal{T}^\pi\eta_1, \Pi_{W_1}\mathcal{T}^\pi\eta_2)$ $< \gamma(s')\bar{d}_\infty(\eta_1, \eta_2) < \gamma_{\text{sup}}\bar{d}_\infty(\eta_1, \eta_2)$. □

Since the application $\Pi_{W_1}\mathcal{T}^\pi$ is also a contraction, we would like to know whether its fixed point is the same than that of $\mathcal{T}^\pi$ and if not, how distant they are from each other. We now introduce the notion of ϵ-approximately Bellman closedness, which measures the distance between the true statistics from the fixed point distribution η_π and the learnt statistics. This was introduced in Rowland et al. [20] and we briefly adapt their results for QGVD.

Definition 4. *A collection of statistics $\theta_1, ..., \theta_N$, together with an imputation strategy Ψ, are said to be ϵ-approximately Bellman closed for a class $\mathcal{M}$ of MDPs, if for each MDP $M = (\mathcal{S}, \mathcal{A}, P, \gamma, c)$ in $\mathcal{M}$ and every policy $\pi \in \mathcal{P}(\mathcal{A})^\mathcal{S}$, we have*

$$\sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} \frac{1}{N}\sum_{i=1}^{N} |\theta_i(\eta_\pi(s,a)) - \hat{\theta}_i(s,a)| \leq \epsilon$$

where $\hat{\theta}_i$ is the learnt value of the true statistic θ_i.

Lemma 4. *Let $\mathcal{M}$ be the class of MDPs such that we define GVDs with the following termination and cumulant functions:*

- $\gamma : \mathcal{S} \to [0, 1)$
- $c : \mathcal{S} \to [-C_{\max}, C_{\max}]$

Then the collection of quantiles $\theta_i = F^{-1}(\frac{2i-1}{2N})$, with the QDRL imputation strategy $\Psi(\theta_1, ..., \theta_N) = \frac{1}{N}\sum_{i=1}^{N} \delta_{\theta_i}$, is ϵ-approximately Bellman closed for $\mathcal{M}$, where $\epsilon = \frac{2C_{\max}(5-2\gamma_{sup})}{(1-\gamma_{sup})^2 N}$

Proof. The result is given in [20] for a constant $\gamma < 1$. Since the function $x : \frac{5-2x}{(1-x)^2}$ is non-decreasing on [0,1], then the result holds as long as $\gamma_{\text{sup}} < 1$. □

Lemma 4 assures us that the average L_1 error between the true quantiles and the learnt quantiles has an upper bound that decreases in $O(\frac{1}{N})$.

3.5 Horde architecture

We will use two types of Horde architectures for our experiments. The first one is very general and allows for any kind of GVFs or GVDs. The second one will be tailored for GVDs, and in our case for QGVDs.

The first Horde's inputs are simply the behaviour policy b and a list of m GVFs and GVDs, which respectively have their own question and answer functions. In this design, each Horde function has its own model and there are no connections between them. They simply try to predict the value function or value distribution separately and we call this framework the **Separate Horde**. Though there is nothing original with this framework, we still want to mark the distinction with the other one.

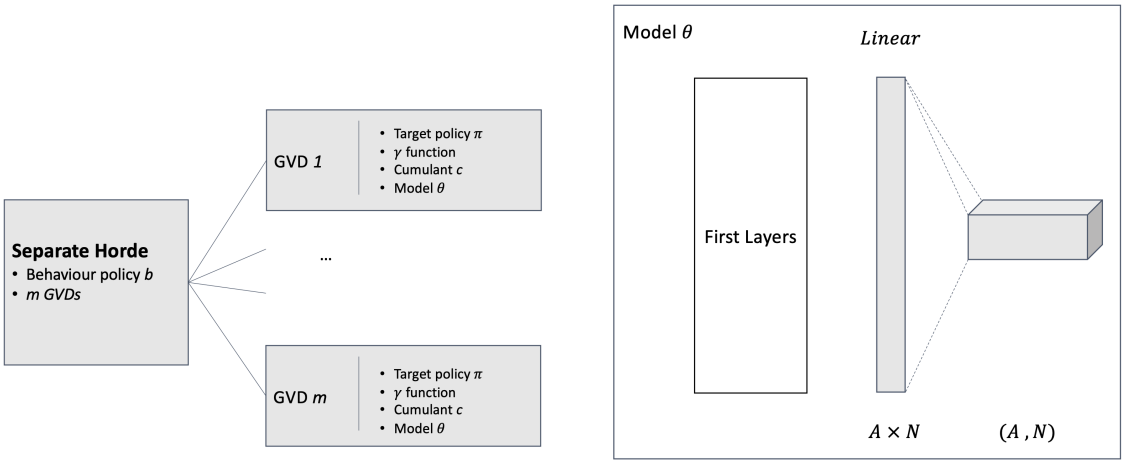
The second architecture is inspired by a remark from Sutton and Barto [25]. The idea here is to have a single Neural Network that is able to output for each state-action pair the return distributions for every GVD. We want to leverage the possibilities of transfer learning between GVDs, so that a GVD defined for an auxiliary task might help in defining the weights for the main task. For simplicity, we consider a single type of method but allowing for multiple ones is possible if we define the associated losses for the Neural Network. This Horde's inputs are therefore the behaviour policy b, a list of m GVDs, and a single neural network. GVDs therefore share the same first layers and only the last layer differentiate between them. We call this architecture the **Unified Horde**.

Both hordes share the following characteristics:

- A horde interacts with its environment by taking a step:
  - The horde is in the current state s'
  - The horde selects the next action a' according to b
  - Each GVD/GVF sequentially "takes a step" by generating a cumulant $C(s, a)$, a termination signal $\gamma(s')$, and performs a learning update according to the update frequency.

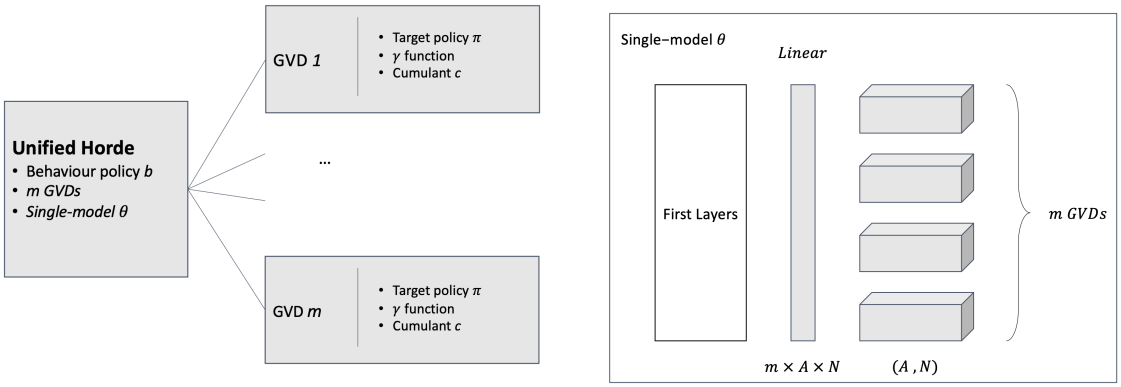
- A horde can be set "on policy" with respect to a particular GVF/GVD. This changes the behaviour policy to a policy based on this value function or distribution.

We present in Figure 3.1 the Separate Horde architecture in the case where all functions are GVDs. Figure 3.2 shows the Unified Horde with its single model θ, whose output is an array of dimensions $(m, |\mathcal{A}|, N)$, where N is the number of quantiles. To compute the quantile huber loss for a particular GVD, we simply slice the output of the Neural Network using the index of the GVD and compute the loss with these θ values. The Unified Model is actually less flexible than its Separate equivalent as all hyper parameters are shared between every GVDs. It is not possible to define a different learning rate for two GVDs, and we cannot change the number of quantiles for a specific value distribution either.



(a) Horde (b) Neural Network model

Figure 3.1: Separate Horde model



(a) Horde (b) Single Neural Network model

Figure 3.2: Unified Horde model

Chapter 4

Mean embedding for Distributional RL

In this chapter we attempt to provide a framework in which we can learn a kernel mean embedding representation of the distribution of the returns. There has been several papers on using Kernel Mean Embedding for Reinforcement Learning. Grünewälder et al. (2012) [8] proposed to learn the transition dynamics of an MDP by estimating the conditional mean embedding $\hat{\mu}(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)\phi(s')} \simeq \frac{1}{n}\sum_{i=1}^{n} \alpha_i(s,a)\phi(s'_i)$. By considering the normalised version of the conditional mean estimate, they show that the Bellman operator is a contraction for the L_∞ norm. Nishiyama et al. (2012) [17], extends this work to Partially Observable MDPs where they adopt a Bayesian approach and provide a kernelised version of the POMDP. Lever et al. (2016) [11] focuses on learning a compressed set of points $c_1, ..., c_m$ to form the estimate of the conditional mean embedding $\frac{1}{m}\sum_{i=1}^{m} \alpha_i(s,a)\phi(c_i)$. In contrast, we focus on the kernel mean embedding of the value distribution and we choose a model-free approach.

4.1 Heuristic approach

Let $k_{\mathcal{S}\times\mathcal{A}}, k_Z$ be symmetric positive-definite kernels on the spaces $\mathcal{S} \times \mathcal{A}$, $\mathbb{R}$. Let ϕ, φ be the associated feature maps and $\mathcal{H}_{\mathcal{A}\times\mathcal{S}}$ and $\mathcal{H}_Z$ the associated feature spaces. In distributional RL, we wish to estimate the distribution $\eta_\pi(s,a)$ for some policy π and for all state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$. We are interested in learning the embedding of this value distribution $\mu_{\eta_\pi^{(s,a)}} = \mu(\eta_\pi^{(s,a)})$. If the kernel kZ is characteristic, then the mapping $\eta_\pi^{(s,a)} \to \mu(\eta_\pi^{(s,a)}) = \mathbb{E}_{z\sim\eta_\pi^{(s,a)}}[\varphi(z)]$ is injective and we get a unique representation of the value distribution in the RKHS.

Let $f \in \mathcal{H}_\mathcal{Z}$ and $\eta \in \mathcal{P}(\mathbb{R})$. Suppose we have access to n returns $z_1, ..., z_n \sim \eta$ and that we would like to compute the expectation $\mathbb{E}_{X\sim P} f(X)$ using the kernel mean embed-

ding μ of η. We can use the kernel mean embedding estimator $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} k_Z(\cdot, z_i)$ to approximate the expectation:

$$\begin{aligned}
\mathbb{E}_{X\sim P} f(X) &= \langle \mu, f \rangle_{\mathcal{H}} \text{ , since} f \in \mathcal{H} \\
&\simeq \langle \hat{\mu}, f \rangle_{\mathcal{H}} \\
&= \langle \frac{1}{n}\sum_{i=1}^{n} k_Z(\cdot, z_i), f \rangle \\
&= \frac{1}{n}\sum_{i=1}^{n} \langle k_Z(\cdot, z_i), f \rangle \text{ , by linearity} \\
&= \frac{1}{n}\sum_{i=1}^{n} f(z_i)
\end{aligned} \tag{4.1}$$

Which yields the classical empirical average estimate. However, if f is not in the RKHS, this does not hold. Kanagawa et al. (2014) [10] showed that with Gaussian Kernels, one could actually recover the moments and probability measures on intervals using samples of an underlying distribution. The result is proved under the assumption that the true and empirical distributions have bounded supports and when $n \to \infty$.

This is an important result, as if we let k_Z be a Gaussian kernel, then we can recover the moments, including the mean, from the return distribution. We thus choose a Gaussian kernel for k_Z.

We now move on to estimate the kernel mean embedding of the conditional distribution $\eta_\pi^{(s,a)}$. We will focus on policy evaluation based on a Monte-Carlo approach, which seems to be the simpler approach and which should be possible given Kanagawa et al. result. Our MDP being episodic, we can generate episodes and compute the observed returns for every visited state-action pair.

An idea would be to adapt the first-visit MC approach by storing samples of returns obtained after the first visit of a state-action pair. That is, if we consider (s, a), let denote $t_{s,a}$ the time where the agent takes the action a in state s for the first time. The associated return $z_{t_{s,a}}$ is a sample from $\eta_\pi^{(s,a)}$. If we simulate n episodes, we can estimate the conditional mean embedding by $\frac{1}{n}\sum_{i=1}^{n} \varphi(z^i_{t_{s,a}})$. However this estimator is heuristic and does not rely on equation (2.32). Notice that this would only work with a discrete and finite state-action space.

We can already see a bottleneck of Conditional Mean Embedding estimation for Reinforcement Learning as the observed samples during a single episode are usually not independent. Throughout an episode, if we denote $(s_1, a_1), ..., (s_T, a_T)$ the successive state-

action pairs of the trajectory and the returns $z_1, ..., z_T$, the triples $(s_1, a_1, z_1), ..., (s_T, a_T, z_T)$ are not independent.

4.2 Naive MC approach

In contrast, the estimator from equation (2.32) is based on the estimates $\hat{C}_{z(sa)}$ and $\hat{C}_{(sa)(sa)}$ and these are used for conditioning by all state-action pairs. One way to obtain a valid estimator would be to generate n episodes, and to select randomly from episode i one triple s^i, a^i, z^i. Having access to the samples $(s^i, a^i, z^i)_{i=1}^n$, let denote
$\Upsilon_{sa} = [k_{\mathcal{S}\times\mathcal{A}}(s^1, a^1), ..., k_{\mathcal{S}\times\mathcal{A}}(s^n, a^n)]$, $\Phi = [k_Z(z^1), ..., k_Z(z^n)]^T$, then we can estimate the conditional mean embedding by

$$\begin{aligned}\hat{\mu}_{\eta_\pi^{(s,a)}} &= \Phi(\Upsilon_{sa}^T \Upsilon_{sa} + n\lambda \mathbf{I}_n)^{-1} \Upsilon_{sa}^T k_{\mathcal{S}\times\mathcal{A}}((s,a),\cdot) \\ &= \Phi\beta\end{aligned} \tag{4.2}$$

This result is more general that the previous one, however it is not very data efficient since it still uses one sample of state-action-return triple per episode. This constraint is imposed because the i.i.d. assumption is required for Equation 2.32. It generalises across the state-action space, and should work with continuous state-action domains. As in every MC method, we can update our estimates only at the end of each episode. However, it is not recommended to systematically update the embedding, as it would otherwise be computationally ineffective for large n. The problematic operation is to compute the inverse of the matrix with an $O(n^3)$ cost.

Keeping only one sample from a whole episode does not seem to be interesting in practice, we suggest instead to treat the whole trajectory as if the samples were independent. In short, we want to compute β, like in Equation (4.2), as if all assumptions for using this result were satisfied and by using the most data that we can. We call this method **Naive MC Mean Embedding**. We proceed in two steps, first we compute the observed returns for every state action pair after the end of each episode according to Algorithm 7. This gives us a vector of returns $\boldsymbol{z}$, last states $\boldsymbol{s}$ and last actions $\boldsymbol{a}$. We then use Equation (4.2) to compute the vector β. During our experiments, we will try to see if we can recover the mean of the distributions using this formula. Since we assumed k_Z to be Gaussian, we actually do not need to specify it further, as we deal directly with returns to compute the mean.

Algorithm 7 On-policy Monte Carlo Returns

Inputs: The policy π to be evaluated, a termination function $\gamma : \mathcal{S} \to [0, 1]$
Output: A vector of returns $\boldsymbol{z}$, and states $\boldsymbol{s}$ and actions $\boldsymbol{a}$
Initialise: $\boldsymbol{z}, \boldsymbol{s}, \boldsymbol{a} \leftarrow$ empty
for *episode in episodes* **do**
  returns, cumulants, gammas $\leftarrow$ empty
  $S \leftarrow S_0$
  for *step in episode* **do**
    Select A according to π
    Take action A
    Observe C, S', and $\gamma_{\text{next}} = \gamma(S')$
    cumulants $\leftarrow$ cumulants + $\{C\}$
    gammas $\leftarrow$ gammas + $\{\gamma_{\text{next}}\}$
    $\boldsymbol{s} \leftarrow \boldsymbol{s} + \{S\}$
    $\boldsymbol{a} \leftarrow \boldsymbol{a} + \{A\}$
  end
  returns $\leftarrow \{C\}$
  $r \leftarrow C$
  for *c, γ in reverse ($cumulants \setminus \{C\}$, $gammas \setminus \{\gamma_{next}\}$)* **do**
    $r \leftarrow c + \gamma r$
    returns $= \{r\}$ + returns
  end
  $\boldsymbol{z} \leftarrow \boldsymbol{z}+$ returns
end

Contrary to the methods developed in the previous chapter, this is on-policy. We thought of changing it into an off-policy version with a Sampling-Importance Resampling procedure (see [18]):

1. Generate n samples z_i from $\eta_b^{(s,a)}$ with fixed (s, a).
2. Compute $\rho_i^{T_i} = \prod_{k=t_i}^{T_i} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$, where t_i and T_i mark the beginning and the end of the cumulants for z_i.
3. Sample B times : z_j^* from $(z_1, ..., z_n)$ with probability $(\tilde{\rho}_1^{T_1}, ..., \tilde{\rho}_n^{T_n})$ where $\tilde{\rho}_i^{T_i} = \frac{\rho_i^{T_i}}{\sum_i \rho_i^{T_i}}$
4. Approximate $\eta_\pi^{(s,a)} = \frac{1}{B} \sum_{j=1}^{j=B} \delta z_j^*$

We however noticed that the effective sample size for the resampled returns was very small and we did not go further with this off-policy scheme.

Chapter 5

Experiments

5.1 Toy Environment

We now describe a toy environment that we will use throughout this chapter. We adapt the 6x10 gridworld environment presented in the Background section and make it more flexible for our experiments. (see Figure 5.1). We keep an episodic MDP, for which episodes terminate when the horde ends up in a terminal state. Terminal states are defined before each experiment through the environment.

Previously, the rewards were non-stochastic and defined as a function of the arrival state of the agent. With the horde architecture, every GVF and GVD have now a different cumulant and γ functions, so we cannot encode them directly in the environment or the horde. Cumulants can also be stochastic and are defined as a function of the last state s, last action a and the current state s'. The termination signal γ is a function of the current state s' only and we allow for deterministic but non constant γ.

This environment is fairly simple as the stochasticity of the transition dynamincs lies in the wall and the obstacles. However, we wanted to have a flexible environment that could imitate the experimental conditions of an agent evolving in a real environment. This setup also makes things easier to consider multiple cumulants at once.

As the grid is 2-dimensional, we refer to states with the coordinates of both axes. In Figure 5.1, the state S is $(5, 0)$, and state A is $(2, 4)$. Alternatively, we will designate S by (50), A by (24) and so on. When not mentioned, the terminal state of the experiment will be $(0, 8)$.

For the state representation, we initially used tile coding for the feature function, motivated by the possibilities of generalisation between states. However this induced another

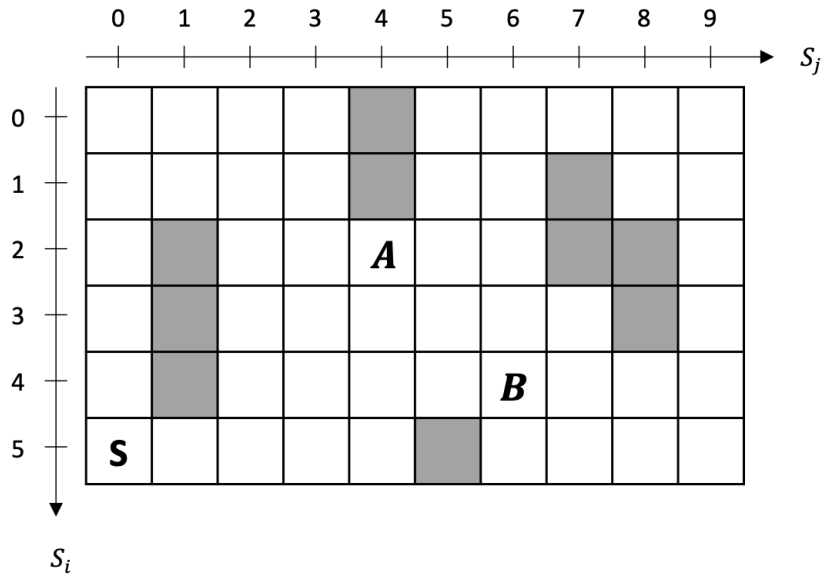


Figure 5.1: The Maze environment used for the toy experiments.

level of approximation that was not really suited for the nature of this environment nor for the purpose of our experiments. We thus focus on a sparse representation:

$$x(s) = x(s_i, s_j) = \begin{pmatrix} 0 \\ \dots \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{pmatrix} \leftarrow s_i \times 10 + s_j \tag{5.1}$$

For GQ and greedy-GQ algorithms, we also need a vector representation of state-action pairs. We simply stack the state vector as follows :

$$x(s,a) = \begin{pmatrix} x(s)\mathbb{1}_{a=\text{up}} \\ x(s)\mathbb{1}_{a=\text{right}} \\ x(s)\mathbb{1}_{a=\text{down}} \\ x(s)\mathbb{1}_{a=\text{left}} \end{pmatrix} \tag{5.2}$$

For QGVD, we use a Neural Network parametrisation to output the quantiles. Since the input state vector is sparse with only one non-zero entry, we cannot use convolutional layers. We build the network with 3 hidden-layers with (128, 64, 32) hidden units. Each of these dense layers has a relu activation. The last part of the network is constituted of a

dense layer of size $|\mathcal{A}| \times N$, that we reshape in an array of size $(|\mathcal{A}|, N)$. See Figure 3.1 b for the complete architecture.

As in DQN and QDRL, we use a prediction network to output the quantiles of the last state-action pair and a target network, used to compute the targets of the updates. For stability, the latter network is updated with the prediction network's weights every F_{update} steps.

Last, during training, every transition (s, a, s', c, γ) is stored in an Experience Replay Buffer. Every F_{replay} steps, we sample B transitions from the buffer and compute the associated targets before computing the loss and taking an optimizer step. We use a batch size B of 32 and an Adam Optimizer with $\epsilon_{\text{ADAM}} = \frac{0.01}{32}$. We will use $\kappa = 1$ and $N = 51$.

5.2 Mean convergence of the algorithms

In this section we would like to demonstrate the mean convergence, i.e. in expectation, of the techniques that we mentioned in Chapter 3. We will look at policy evaluation and control in fairly simple settings to make sure that our algorithms work properly. It is also a way of comparing GVF and GVD techniques as we will later on focus on GVD. We will study only one Horde function at a time, since the goal is to observe convergence on known settings.

5.2.1 Off-policy evaluation

For policy evaluation, we work with a sparse Bernouilli cumulant function, i.e. a cumulant function that outputs zero for all current states but the objective state S_G, where the cumulant is bernouilli. It is defined as follows:

$$c_{\text{Bernouilli}}(S, A, S') = \begin{cases} 0 & \text{if } S' \neq S_G \\ 20 \times X \text{ with } X \sim \mathcal{B}(1/2) & \text{if } S' = S_G \end{cases} \tag{5.3}$$

The γ function will be equal to 0.95 except when the current state is S_G, where it is 0. This way, we define GVFs just as in usual settings with a terminal objective state. The only difference is that, when the horde reaches S_G, the episode does not end right away. Yet, the γ function allows the horde to consider the sum of discounted cumulants until we reach S_G. Transitions that occur after that are still used for learning.

We use a behaviour policy that selects actions uniformly at random, so that we can explore efficiently the state space. We set the objective state $S_G = (4, 8)$.

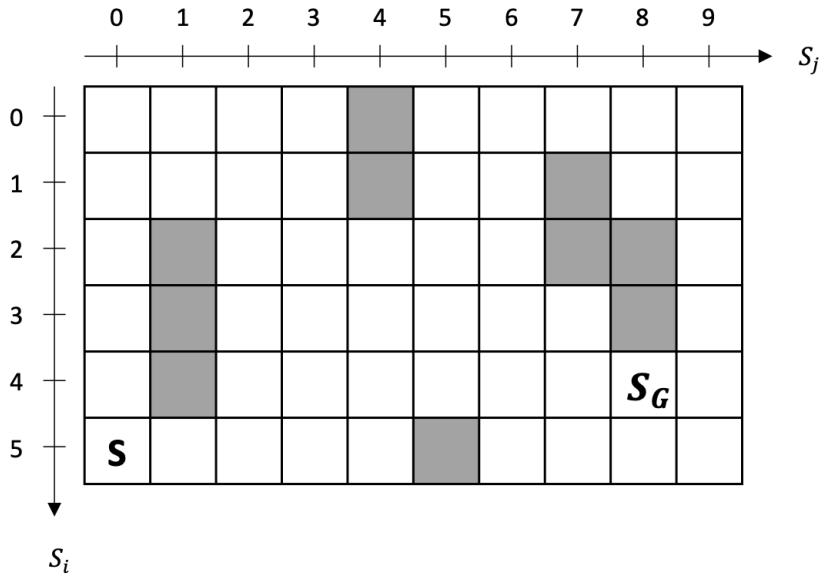


Figure 5.2: Objective state for Off-policy evaluation

We will use the Root Mean Square Value Error (RMSVE) metric to compare in average how far the algorithms' value estimates are from the theoretical values. The measure μ is in practice the frequency of being in a state under the behaviour policy b. We compute $\hat{v}$ based on the q values and Equation (2.5).

$$\text{RMSVE}^2 = \sum_{s \in \mathcal{S}} \mu(s)(v_\pi(s) - \hat{v}(s))^2 \tag{5.4}$$

As we want to easily compare the algorithm results to theoretical state values, we choose a target policy π_{right} that selects always the action 'going right'. This migth appear to be too simple, however it is relevant for two reasons:

1. First, computing the state values in this case is easy to do on paper, as every state which is not directly to the left of S_G will be zero. As for the others, we can compute them with $v_{\pi_{\text{right}}}((4,7)) = 10$ and $\gamma = 0.95$. We thus have a clear way of comparing our algorithms to the ground truth.

2. This question is actually relevant for off-policy settings and particularly for a horde framework. We could not answer the question: "what is the value of being in this state if I keep going right for these cumulant and γ functions" using on-policy methods. We can also imagine a horde that gets 4 GVFs as inputs, with the same cumulants and termination signals, but each with a policy that goes only in one direction.

We run the experiment for 1000 episodes and compute the RMSVE at the end of each

episode. We average the results over 10 runs for GQ(λ) and 3 runs for the Average-Loss QGVD. We use $\lambda = 0.1$ and $\alpha_h = 0.1 \times \alpha$ for GQ. For QGVD, we train with an experience replay every 5 steps, update the target network every 5000 steps.

We plot the RMSVE for different learning rates in Figure 5.3 and Figure 5.4. We can notice that the learning rate needs to be chosen with due consideration. For GQ, if it is too low, learning does not occur and for values such as $\alpha = 0.1$, the RMSVE does not look very stable over time. For QGVD, a too high learning rate seems to introduce instability in the convergence process and we should favour learning rate of 1×10^{-4} and 5×10^{-5} but not higher.

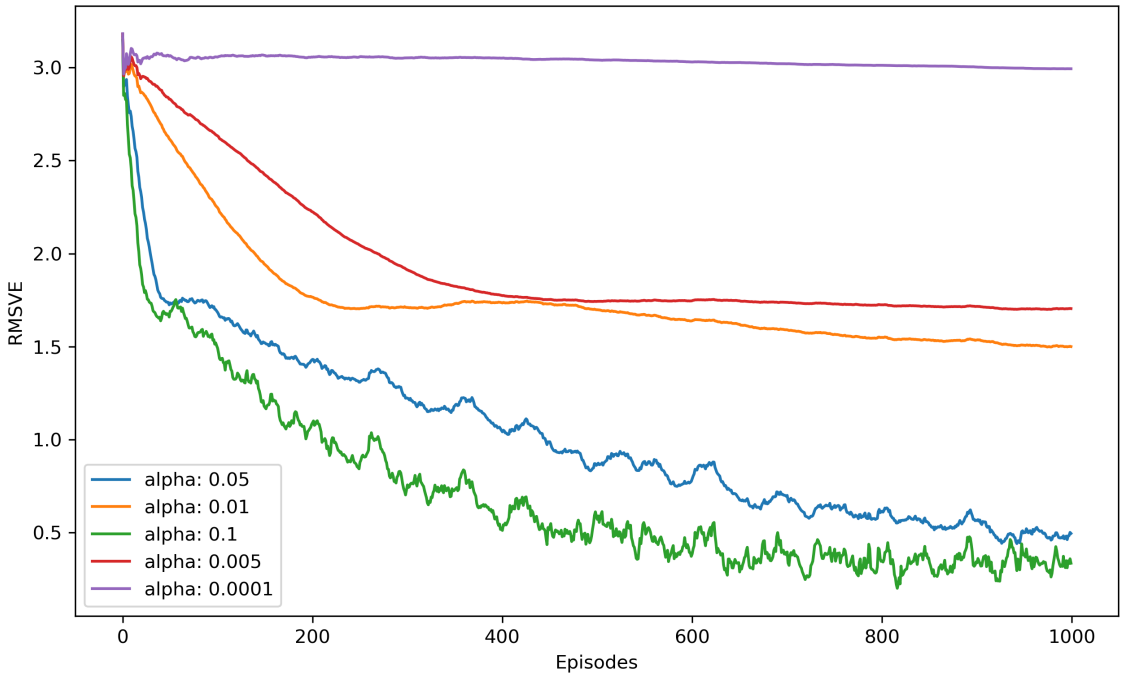


Figure 5.3: RMSVE of values obtaied with GQ(λ)

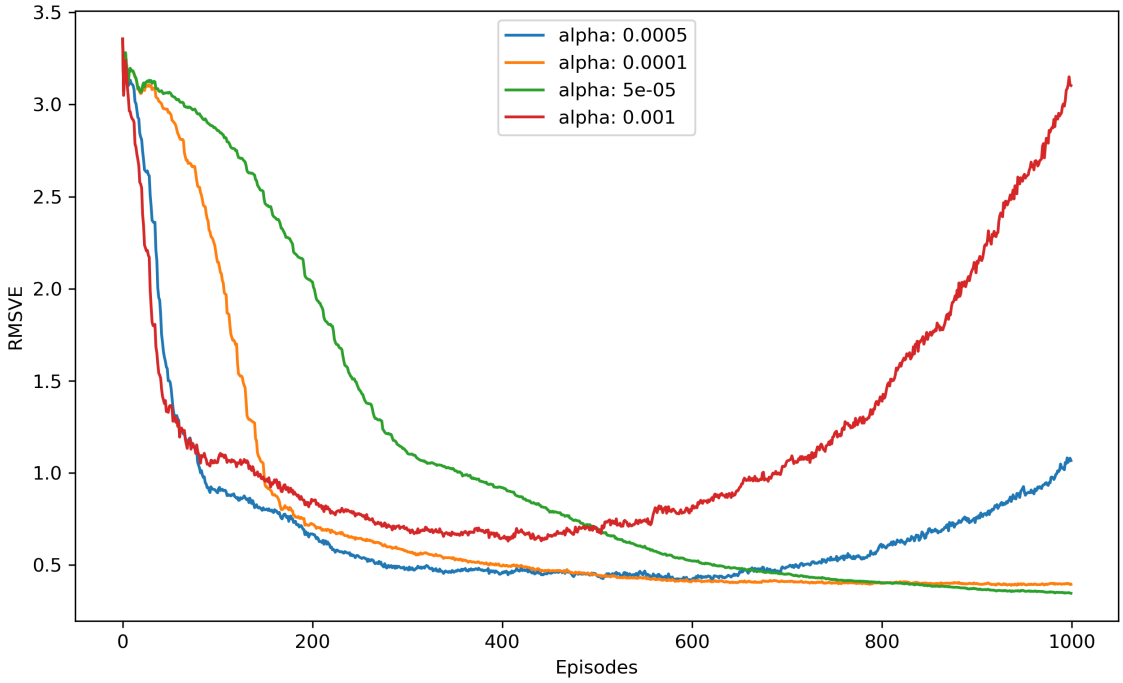


Figure 5.4: RMSVE of values obtained with Average Loss QGVD

We also plot the corresponding q values obtained at the end of a single run in Figure 5.5 and Figure 5.6. The main difference between QGVD and GQ(λ) in this case, is that q values that should be exactly zero are equal to zero in the case of GQ, and approximately 0 with QGVD. This is surely due of the weight initialization.

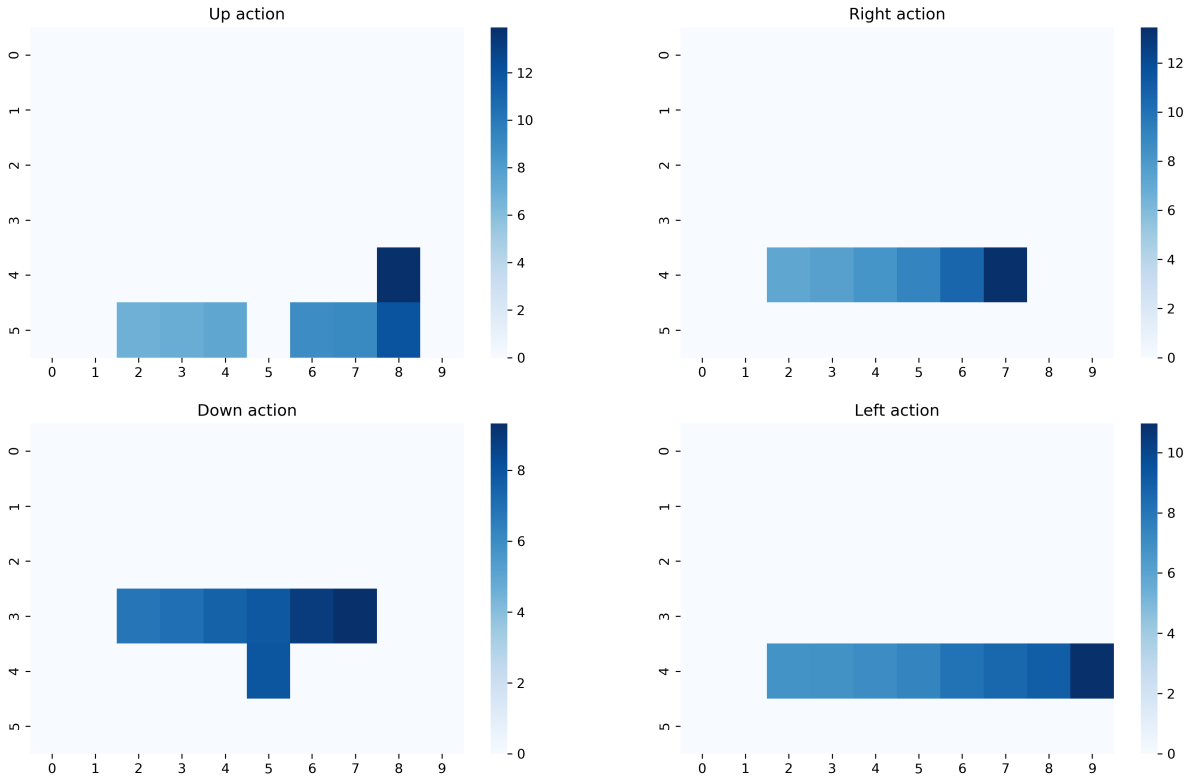


Figure 5.5: q values obtained with GQ(λ), $\alpha = 0.1$

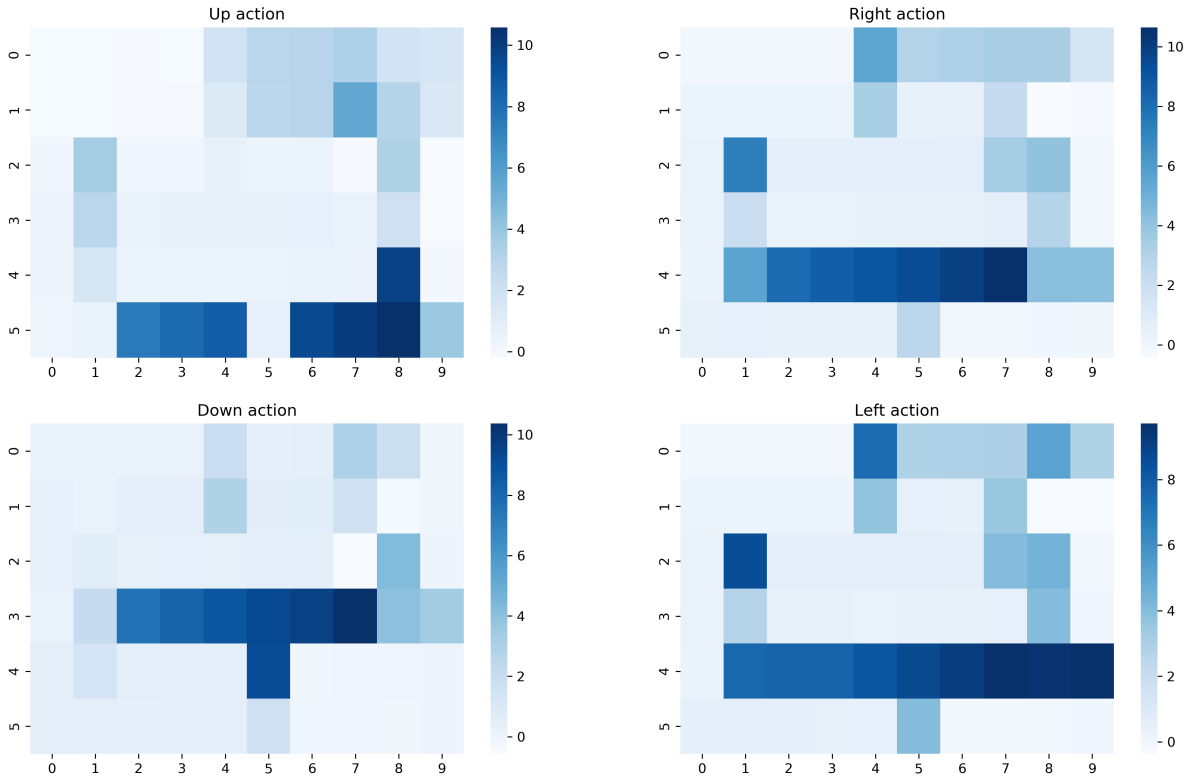


Figure 5.6: q values obtained with QGVD, $\alpha = 5 \times 10^{-5}$

5.2.2 Off-policy control

For control, we use a sparse Gaussian cumulant function defined as follows:

$$c_{\text{Gaussian}}(S, A, S') = \begin{cases} 0 & \text{if } S' \neq S_G \\ Z \text{ with } Z \sim \mathcal{N}(\mu, \sigma^2) & \text{if } S' = S_G \end{cases} \tag{5.5}$$

We set $\mu = 10$, $\sigma = 2$, and $S_G = (2, 4)$. We choose the same termination function than for policy evaluation. We again use the RMSVE as an indicator of mean convergence. We compute $v_{\pi^*}(s)$ using the shortest path $d(s, S_G)$ from s to S_G: $v_{\pi^*}(s) = (0.95)^{d-1} \times 10$.

We run the experiment for 1000 episodes and plot the averaged RMSVE for greedy-GQ and Q-learning QGVD in Figure 5.7 and Figure 5.8. We observe convergence in both cases.

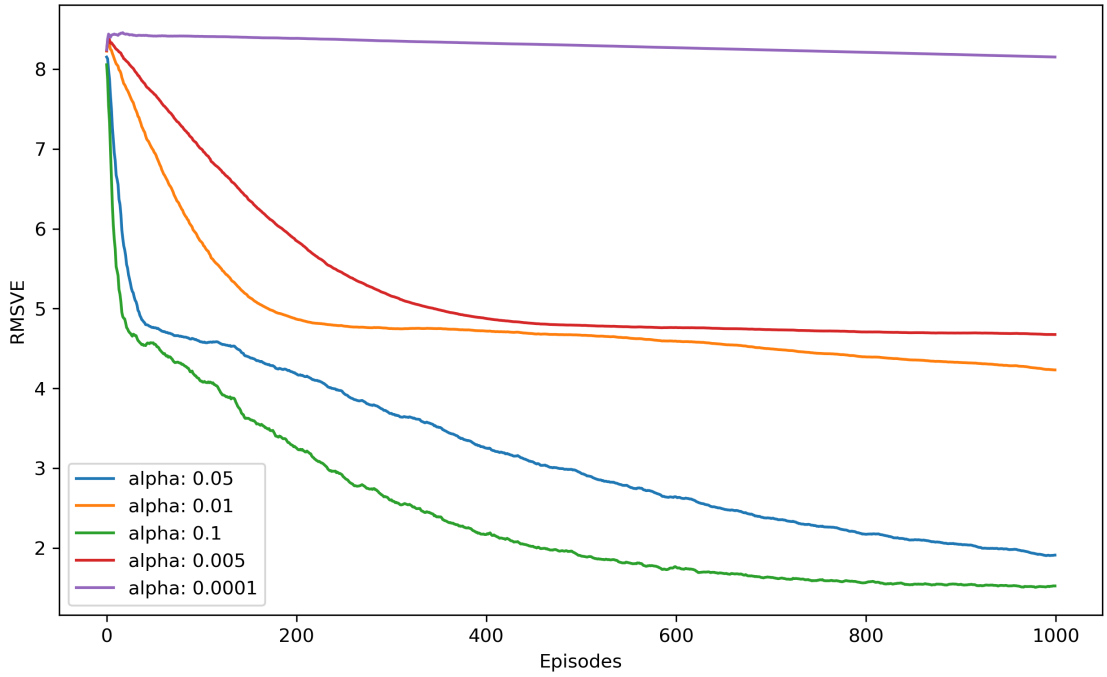


Figure 5.7: RMSVE of values obtained with greedy GQ

5.2.3 On-policy evaluation with Naive Mean Embedding

We present in this section the experimental results obtained with Naive MC Mean Embedding. The objective is to recover the q values using the β weights: $q(s,a) = \sum_{i=1}^{n} \beta_i(s,a) z_i$. We use the sparse Gaussian cumulant function from previous section and the same termination function as before. The policy selects action uniformly at random. We set $S_G = (2,4)$.

We employ the following vector representation of a state action pair: $x(s,a) = \begin{pmatrix} s_i \\ s_j \\ 2(\mathbb{1}_{a=\text{up}} - \mathbb{1}_{a=\text{down}}) \\ 2(\mathbb{1}_{a=\text{right}} - \mathbb{1}_{a=\text{left}}) \end{pmatrix}$.

In doing so, if $s = s'$, $x(s,a) - x(s,a') = 0$ iff $a = a'$. Besides, the norm of the difference will be larger if $a =$ up, $a' =$ down than if $a =$ up, $a' =$ left, which is reasonable.

We then define the kernel $k_{\mathcal{S}\times\mathcal{A}}(\cdot,\cdot)$ as follows:

$$k_{\mathcal{S}\times\mathcal{A}}((s,a),(s',a')) = \exp\left(\frac{\|x(s,a) - x(s',a')\|_2^2}{\sigma^2}\right) \tag{5.6}$$

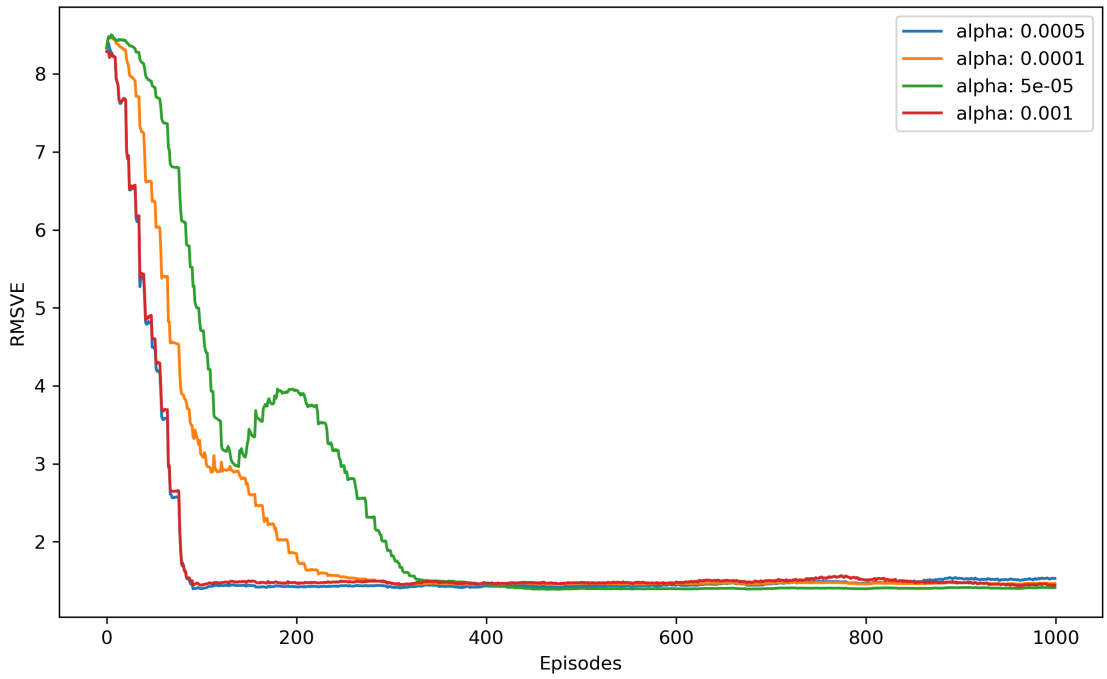


Figure 5.8: RMSVE of values obtained with Q-Learning QGVD

Computing the inverse matrix represents a prohibitive cost, as a result we run the experiment for only 20 episodes to get a vector of returns of moderate size. We choose $\lambda = \frac{1}{10n}$, where n is the total number of steps of the experiment. We sweep over several values of $\sigma \in \{0.5, 1, 2, 5\}$ and we show the associated RMSVE in Table 5.1. The reference is the returns obtained when running Algorithm 7 for 1000 episodes.

σ	RMSVE
0.5	0.257
1	0.258
2	0.258
5	0.349

Table 5.1: RMSVE of the state values obtained with Naive MC Embedding for several values of σ. The reference is the empirical return distribution observed for 1000 episodes.

We also plot q values for $\sigma = 0.5$ and $\sigma = 5$ in Figure 5.9 and Figure 5.10 . Notice that when we increase σ, the q values become more locally uniform as the kernel between two state-action pairs tends towards 1 when $\sigma \to \infty$.

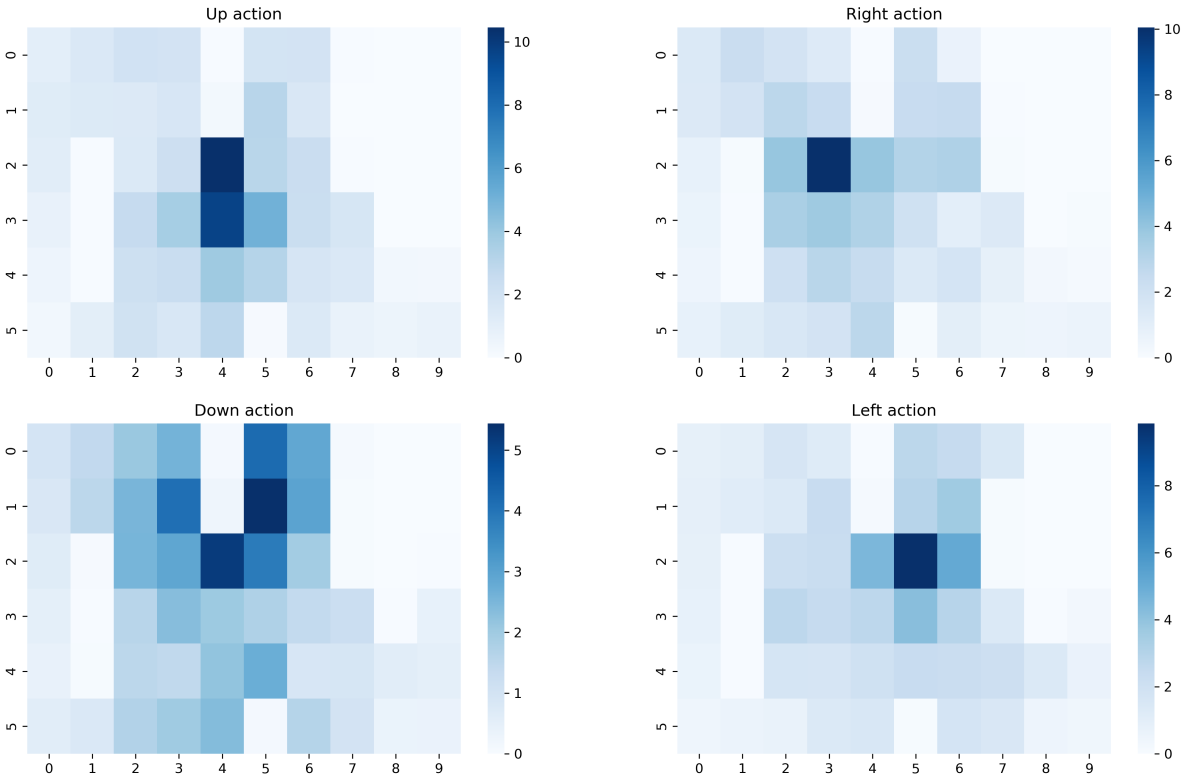


Figure 5.9: q values obtained with Naive MC Mean Embedding for $\sigma = 0.5$ after 20 episodes.

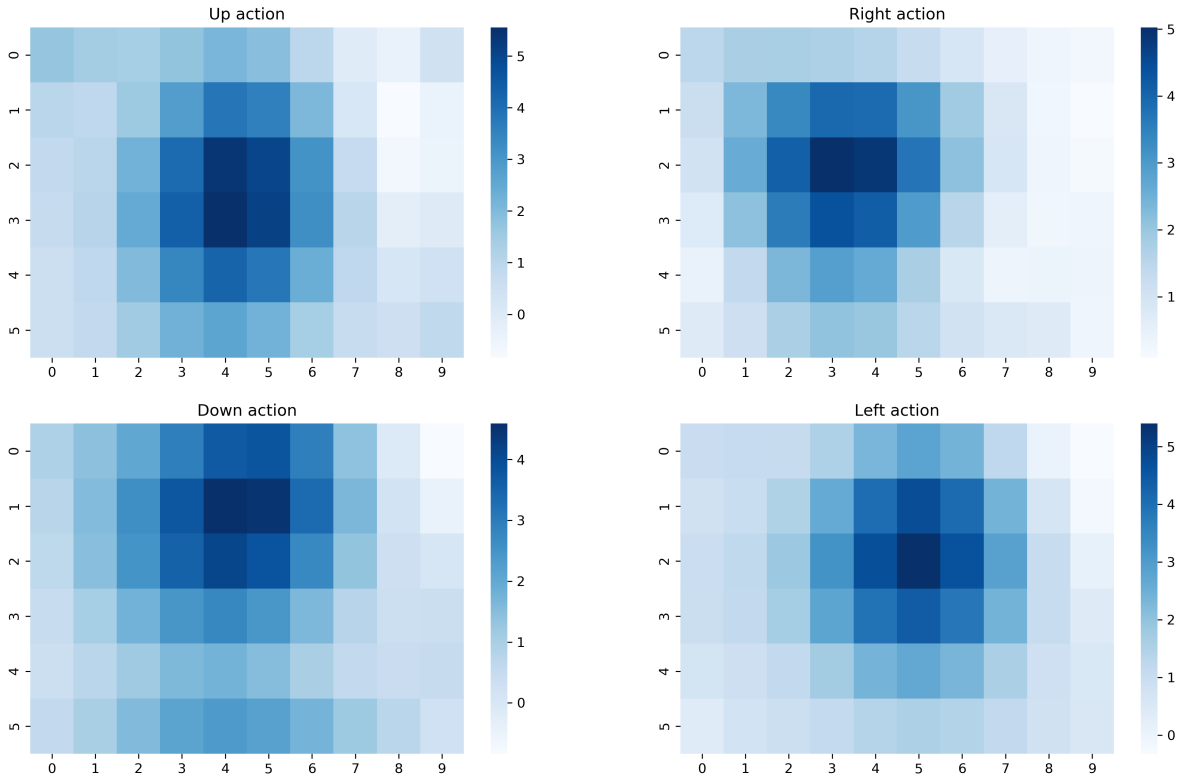


Figure 5.10: q values obtained with Naive MC Mean Embedding for $\sigma = 5$ after 20 episodes.

By analysing the β weights for a pair (s, a), we observe that for $\sigma = 0.5$ and $\sigma = 1$, the only non-negligible weights are those with state actions that match exactly the pair. This means that only the returns observed with the same pair contribute to its q value.

We noted that for larger values of σ, this is not the case anymore, and non-negligible β weights get attributed to state-action pairs that are close to (s, a). This enables to get an estimate for a state action pair that might not have been visited during training.

5.3 Distributional Convergence

As QDRL is supposed to find an approximate distribution which minimises the wasserstein distance with the empirical distribution, we generate return samples for every state-action using Algorithm 7. This will be the reference to which we compare the θ values obtained with our method. We generate returns with an experiment of 1000 episodes.

This is an on-policy method, and it is generally not suited for policies that do not explore all the state space properly. However, we will mainly look at state-actions that separate the starting state S of the episode from the objective state S_G, and this should not be a problem as long as we visit these states and the objective state often.

5.3.1 Off-policy evaluation

We use again the sparse Gaussian cumulant and the same γ function. We set $S_G = (2, 4)$. While training, we track the wasserstein distance between the current $\theta(s, a)$ values and the returns obtained with the MC method for a list of state-actions : (23, right), (33, up), (43, up), (51, right), (50, right) (see Figure 5.11). We use a learning rate of 5×10^{-5}, we replay every 5 steps and update the target network every 500 steps.

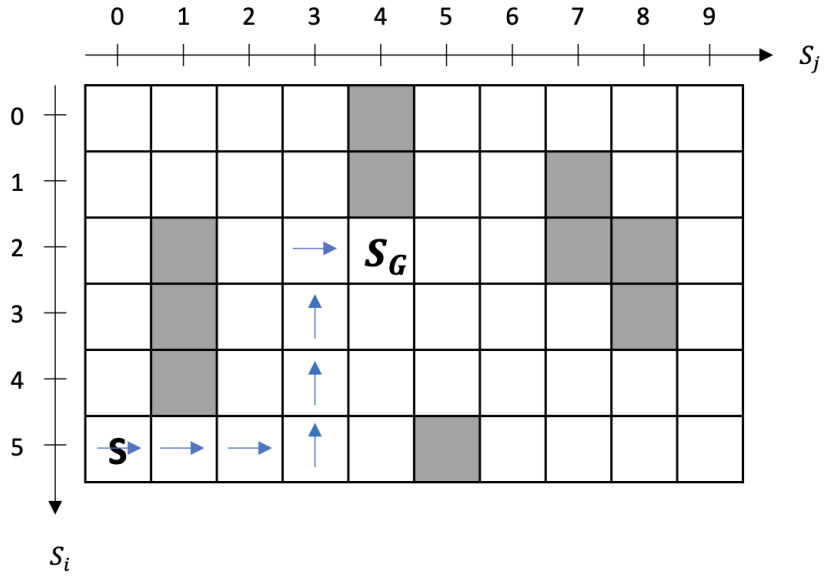


Figure 5.11: Goal state for the off-policy evaluation. The blue arrows represent the state action pairs for which we track the wasserstein distance with the monte carlo samples. The deterministic policy will select the arrow actions that lead to S_G starting from S.

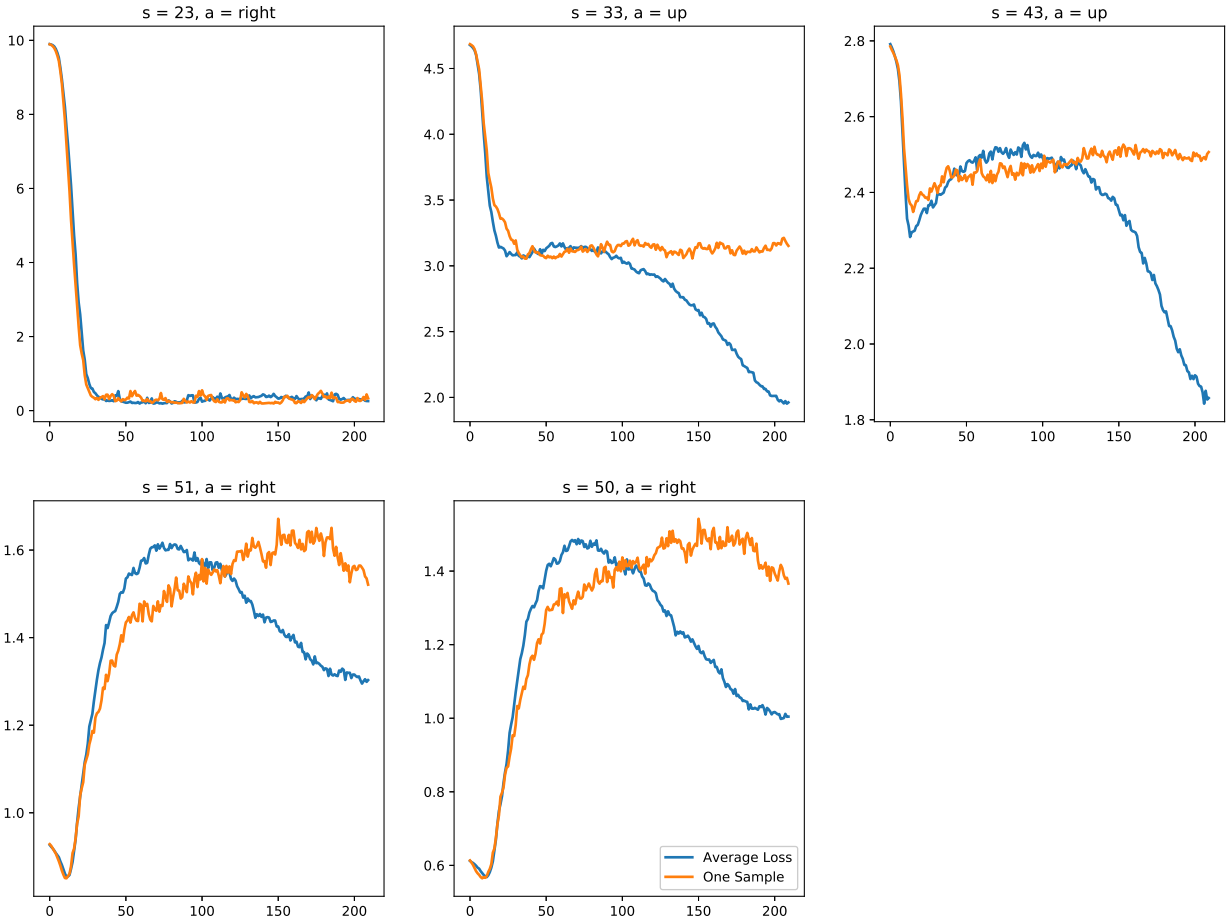


Figure 5.12: Policy evaluation with a Dirichlet policy. Wasserstein distance between the quantile approximation and the Monte Carlo samples every 1000 steps.

We start the analysis with a fixed Dirichlet policy $\pi_{\text{dirichlet}}(s, \cdot) \sim \text{Dirichlet}(\alpha = (10, 10, 10, 10))$. The wasserstein distance every 1000 steps is shown for a 500-episode run with Average-Loss QGVD and One-Sample QGVD in Figure 5.12. We notice that, though the state-actions that are close to the objective state converge in Wasserstein distance, it seems to take more training time for further state actions such as (50, right) or (51, right). We suspect that the training in such case is expected to be very slow as Deep Reinforcement Learning is known to be very sample inefficient. Besides, the stochasticity induced by the policy might complicate learning. Nevertheless, on this example we can notice that the wasserstein distance obtained with the Average-Loss algorithm seems to decrease more steadily than for the One-Sample variant.

We now look at a semi-deterministic policy that leads straight to S_G from S (blue arrows in Figure 5.11) and that selects actions uniformly at random in other states. We expect learning to be faster for the state-action distributions of our track list, as the theta values should propagate more directly along the path. We plot the Wasserstein distance for both algorithms in Figure 5.13 and we notice that the convergence is indeed faster than previously. Qualitatively, we can witness that the wasserstein distance oscillates less for the Average Loss algorithm. Nevertheless, for $(51, \text{right})$, and $(50, \text{right})$, the metric for the One Sample method is lower at the end of the experiment, which indicates that there is no clear cut between the two methods. It might sound surprising to have different results for state-action pairs along the path. However, the weights are global and are affected by transitions with a stochastic policy.

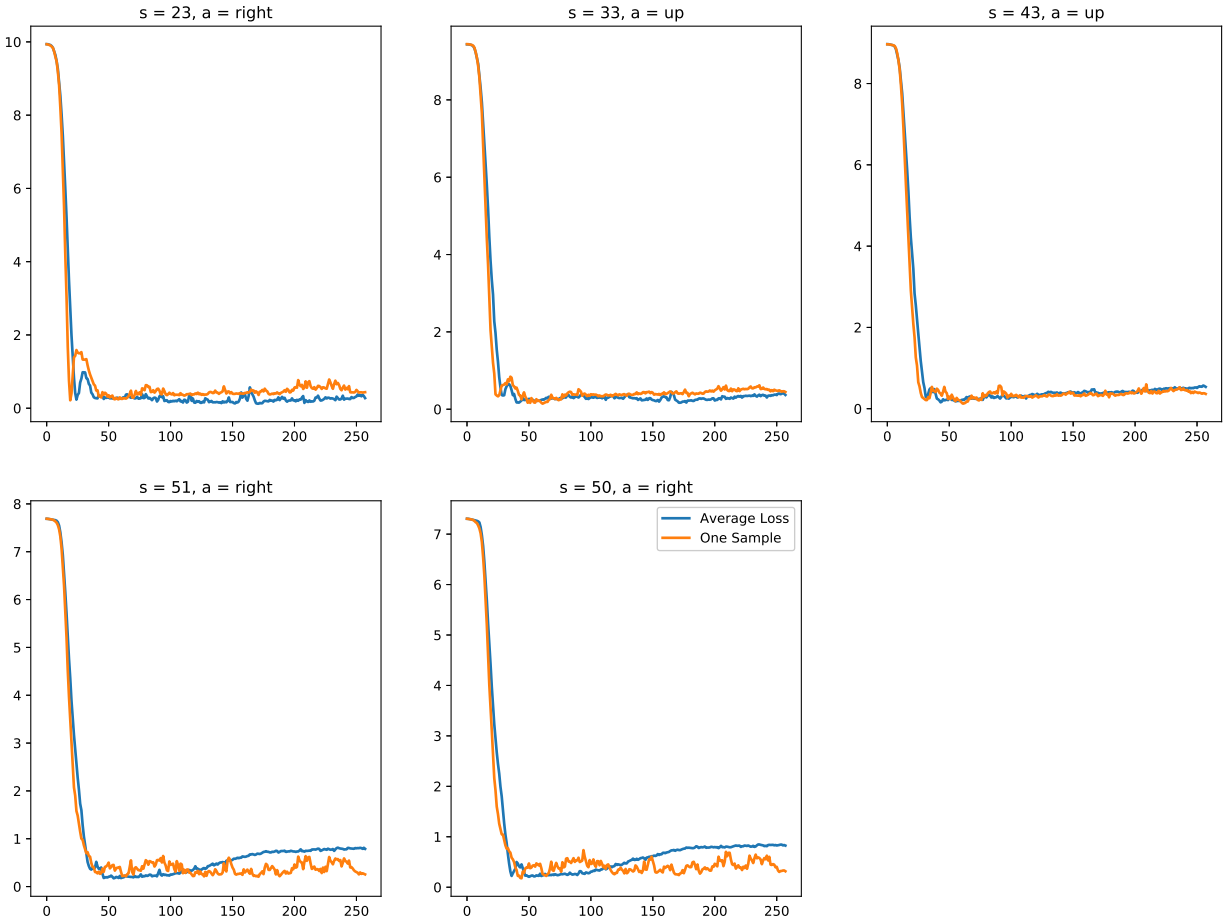


Figure 5.13: Policy evaluation with a semi-deterministic policy. Wasserstein distance between the quantile approximation and the Monte Carlo samples every 1000 steps.

Finally, as the previous policy reduced a lot of stochasticity, we replace the deterministic actions of the previous policy by a stochastic counterpart, where the probability of advancing along the path from S to S_G is 0.85 and the probability for taking one of the other actions is 0.05. In Figure 5.14 we obtain comparable results than with the semi-deterministic policy. The One Sample method is actually doing better for this policy on this list of state. Our interpretation is that One Sample performs best when the policy is deterministic or semi-deterministic.

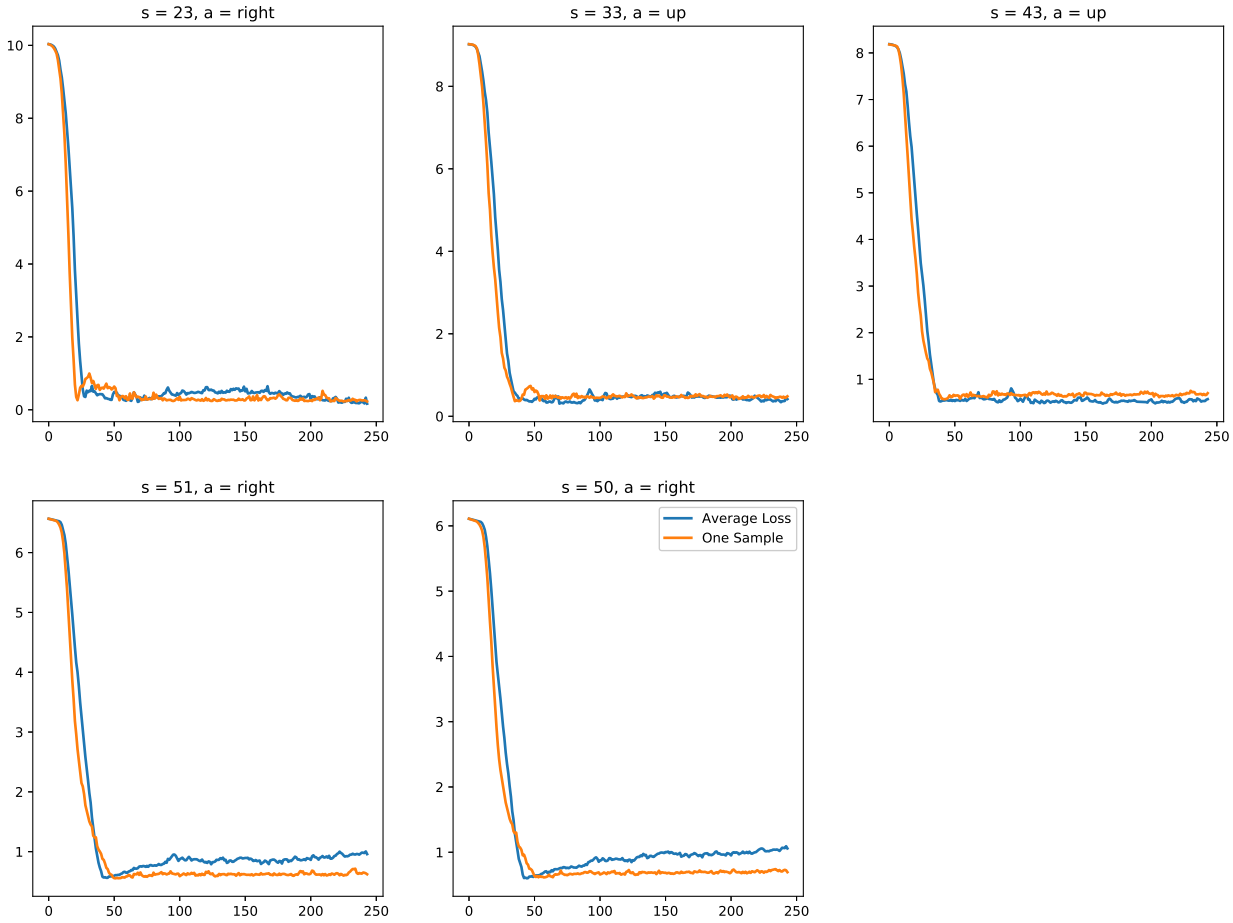


Figure 5.14: Policy evaluation with the third policy. Wasserstein distance between the quantile approximation and the Monte Carlo samples every 1000 steps.

5.3.2 Off-policy control

We now turn to a control setting. We keep the same parameters as for evaluation but run the experiment for 1000 episodes. We use the risk neutral version of Q-Learning, with $\beta = 0$. We show the Wassertein distance while training in Figure 5.15. The distance for all state-actions decrease gradually in the beginning and more smoothly once the policy is somewhat fixed. Note that in the control setting, as the policy is deterministic, the propagation of the distributions from the objective state to further states is actually easier.

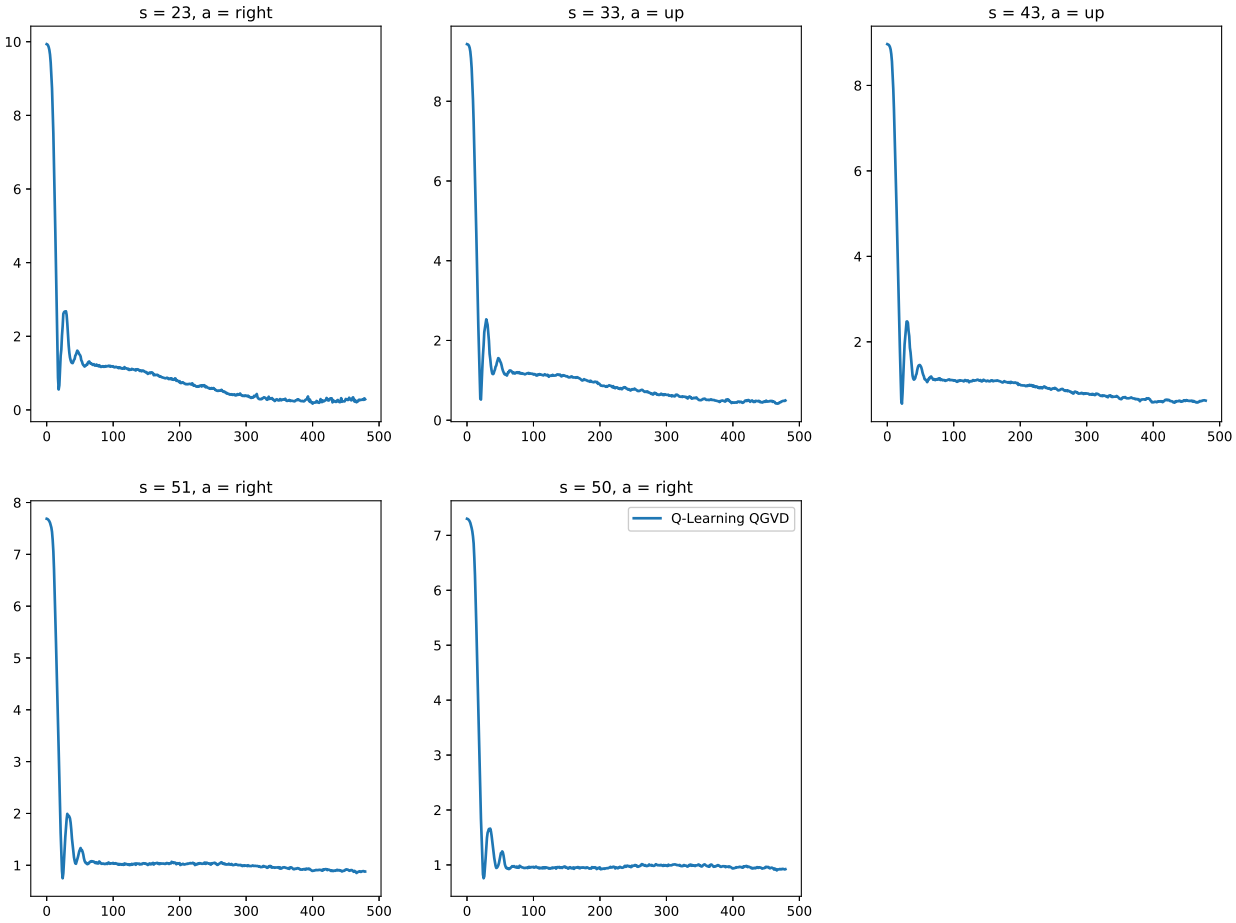


Figure 5.15: Control setting. Wasserstein distance between the quantile approximation and the Monte Carlo samples every 1000 steps.

5.4 Risk-sensitive Q-Learning

For this section, we use a sparse cumulant function with two objective states A and B (see Figure 5.1).

$$c_{\text{Two Gaussians}}(S, A, S') = \begin{cases} 0 & \text{if } S' \neq (A \text{and } B) \\ Z_1 \text{ with } Z_1 \sim \mathcal{N}(\mu, \sigma_1^2) & \text{if } S' = A \\ Z_2 \text{ with } Z_2 \sim \mathcal{N}(\mu, \sigma_2^2) & \text{if } S' = B \end{cases} \tag{5.7}$$

We set $\mu = 10$, $\sigma_1 = 2$, $\sigma_2 = 5$, $\gamma(A) = \gamma(B) = 0$, and $\gamma = 0.95$ otherwise. This is a way to consider discounted cumulants until we either get to A or B. A and B are located at the same distance from the starting state.

Both distributions have the same mean but present different risk levels. A risk-averse agent would tend to go to state A while a risk-seeking one would preferably choose state B. We want to verify experimentally that our risk-sensitive Q-Learning is consistent with this a priori.

We start with an offline control setting, where the terminal state is in $(0, 8)$ and where the behaviour policy selects actions uniformly at random. We run the experiment for 500 episodes with $F_{\text{replay}} = 5$ and $F_{\text{update}} = 500$ for $\beta \in .\{-3, 3\}$.

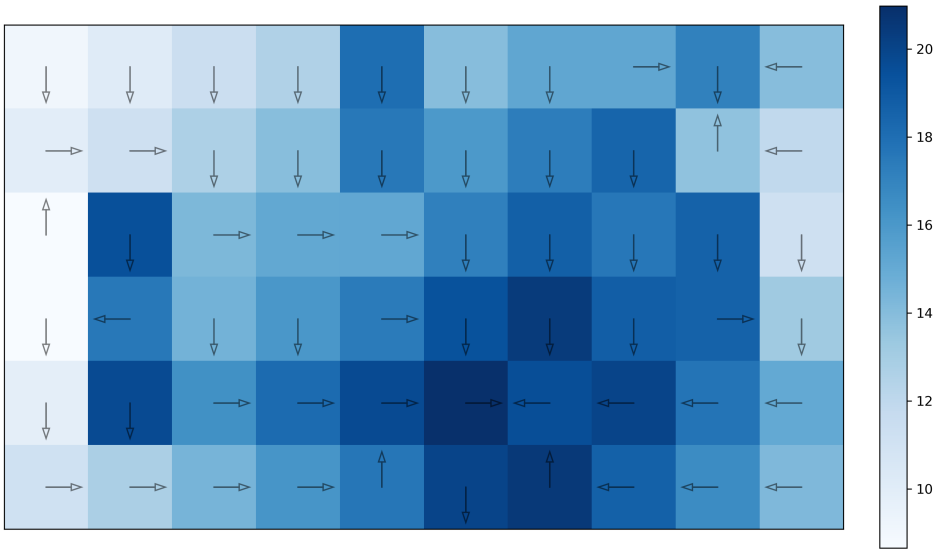


Figure 5.16: State values and deterministic policy obtained with $\beta = 3$

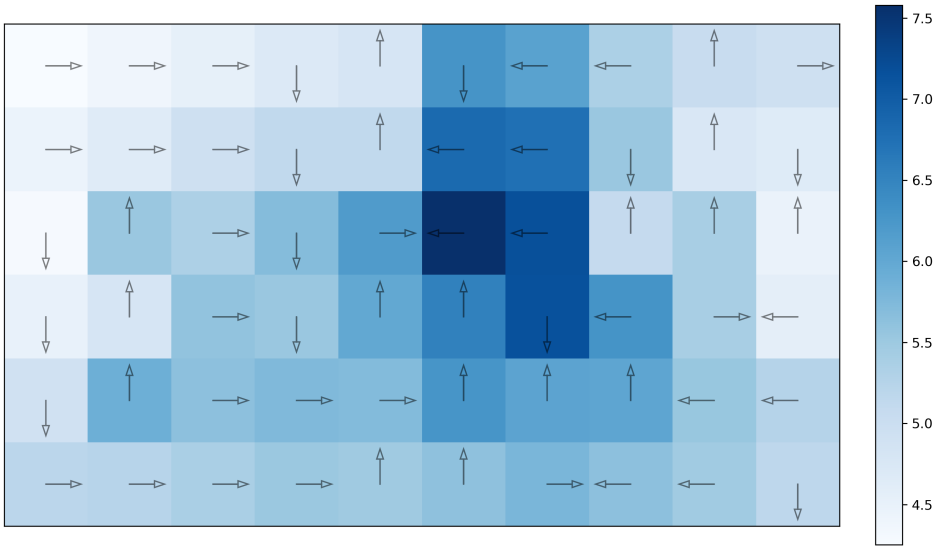


Figure 5.17: State values and deterministic policy obtained with $\beta = -3$

We plot the v_β values as well as the policy derived from the q_β values in Figure 5.16 and Figure 5.17. We conclude that the risk sensitiveness works as expected since π_3 leads to A and π_{-3} leads to B.

Furthermore, we are interested in knowing whether the agent can discriminate between the two objectives online. We set A and B to be terminal states to avoid never-ending episodes. We use an ϵ-greedy behaviour policy with respect to q^*_β, with an ϵ parameter that decays over episodes. For episode i:

$$\epsilon(i) = \frac{1}{1 + \sqrt{\lfloor \frac{i}{25} \rfloor}} \tag{5.8}$$

We run the experiment for 500 episodes, for which ϵ varies from 1 to 0.2. This should enable a good exploration of the two states. We put the focus on exploration over exploitation so that the horde can explore sufficiently well the two objectives to discriminate between them. We update the behaviour policy with the target network, so that the updates are less frequent but generate a more stable behaviour. We average the results over 10 runs and plot the number of steps per episode in Figure 5.20, and which states the horde visits during the last 10 episodes of a run in Figure 5.18 and Figure 5.19.

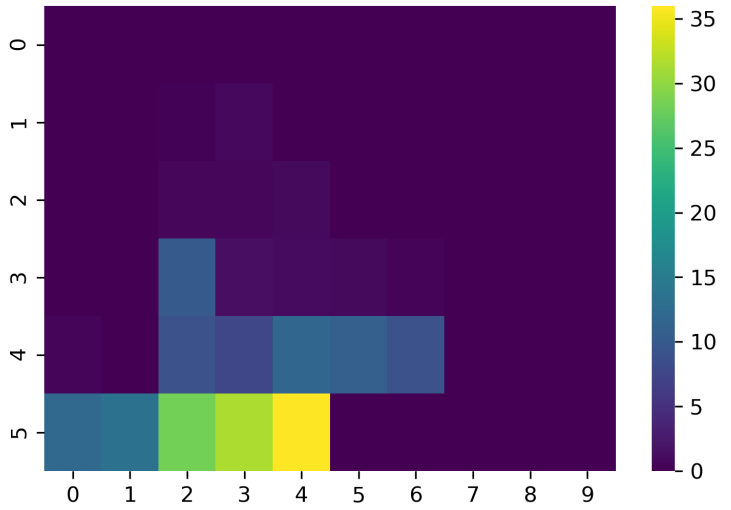


Figure 5.18: Average state visits during the last 10 episodes of a run with $\beta = 3$

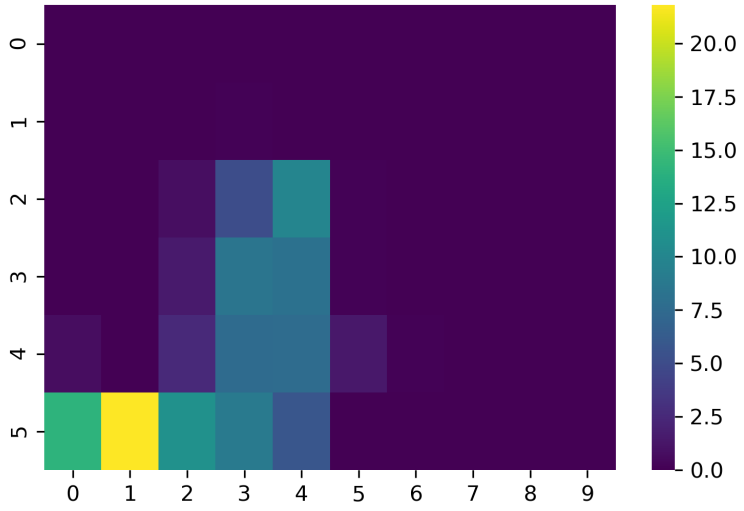
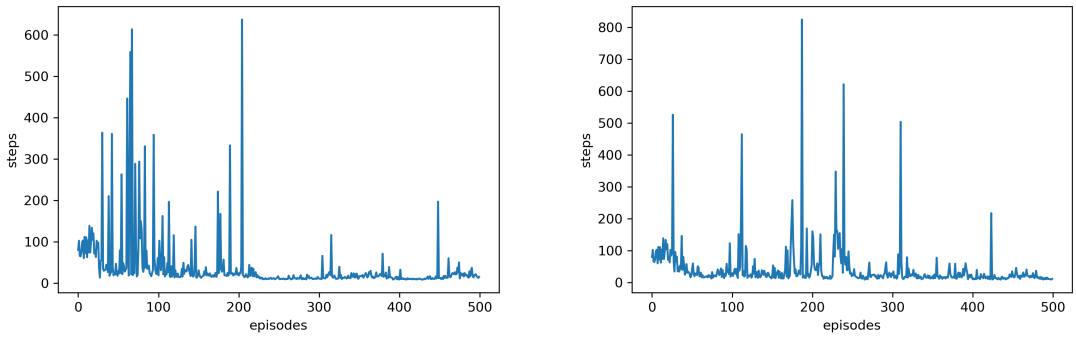


Figure 5.19: Average state visits during the last 10 episodes of a run with $\beta = -3$



(a) $\beta = 3$ (b) $\beta = -3$

Figure 5.20: Average steps per episode throughout a run with online risk-sensitive Q-Learning QGVD

5.5 Multiple GVDs

In this section, we will demonstrate that the Unified Horde is able to learn about multiple GVDs in parallel. We actually used the Separate Horde architecture for previous experiments, and used two GVDs for the Average Loss vs One Sample comparison. For efficiency, we cover the case of two GVDs.

The first GVD that we use is the same as for Off-policy evaluation with the sparse Gaussian cumulant and the semi-deterministic policy of section 5.3.1. The second one is used to detect the walls of the maze. Indeed, the Wall detector demon detects a wall with probability p if the horde's current state is next to a wall and if the horde was in motion. We set $p = 0.8$ and the target policy selects actions uniformly at random. Let S_{Walls} be the list of states adjacent to the perimeter of the maze, namely $(0,0), ...(0,9), (1,9), ...(5,9), ...$ We define the following cumulant and termination functions:

$$c_{\text{Wall Detector}}(S, A, S') = \begin{cases} 0 & \text{if } S' \notin S_{\text{Walls}} \\ X \text{ with } X \sim \text{Bernouilli}(p) & \text{if } S' \in S_{\text{Walls}} \text{ and } S' \neq S \end{cases} \tag{5.9}$$

$$\gamma_{\text{Wall Detector}}(S, A, S') = \begin{cases} 0.1 & \text{if } S' \notin S_{\text{Walls}} \\ 0.95 & \text{if } S' \in S_{\text{Walls}} \text{ and } S' \neq S \end{cases} \tag{5.10}$$

We run the experiment for 1000 episodes, replay every 5 steps, update the target network every 500 steps and the learning rate is 5×10^{-5}. We use the Average Loss algorithm. We show in Figure 5.21 the wasserstein distance for return distributions for both GVDs while training. We conclude that the method is effectively converging, and especially fast for

the wall detector GVD. We also plot the state-action values in Figure 5.22 and Figure 5.23 to get a sense of the results in Expectation.

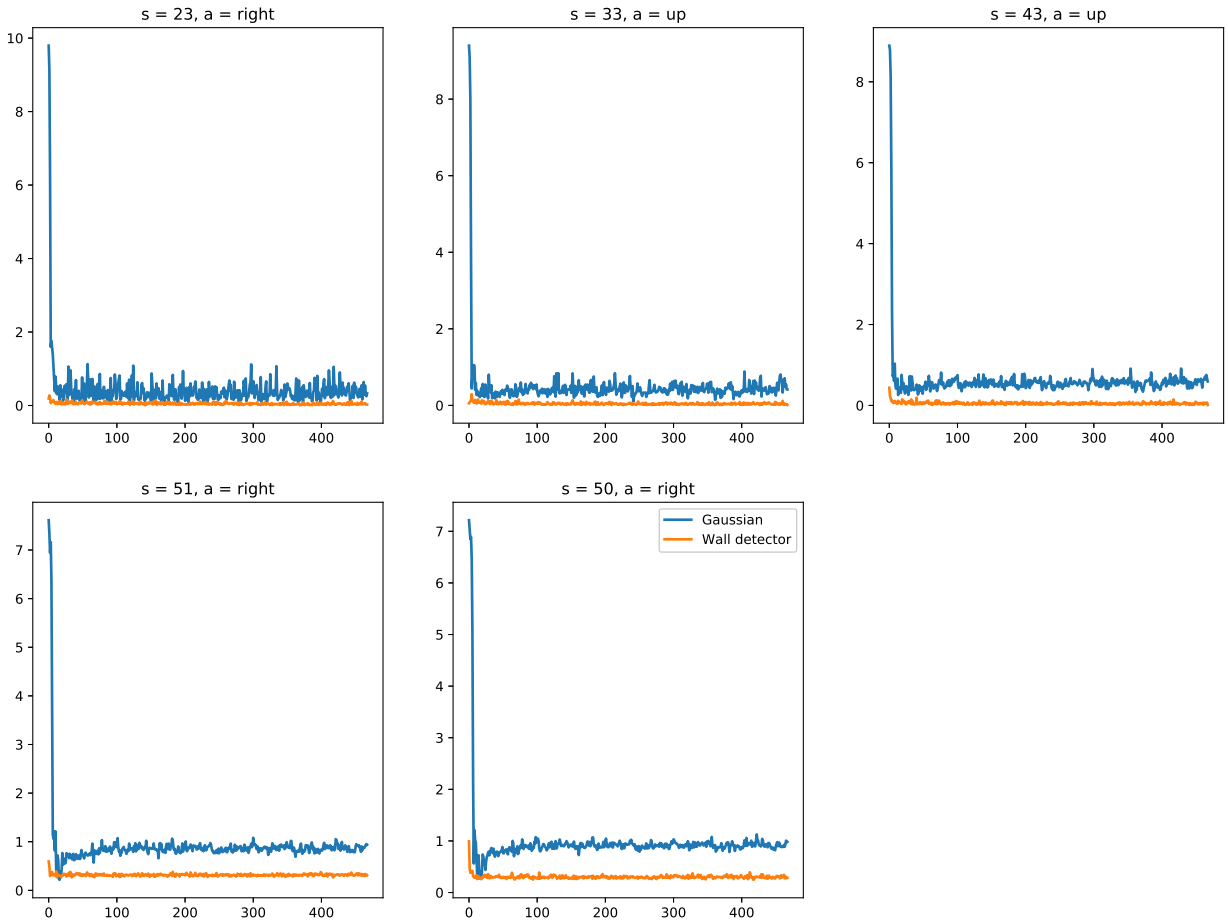


Figure 5.21: Unified Horde with two evaluation GVDs : sparse Gaussian cumulant with semi-deterministic policy and wall detector GVD with uniform policy. Wasserstein distance between the quantile approximation and the Monte Carlo samples every 1000 steps. Average Loss QGVD is used.

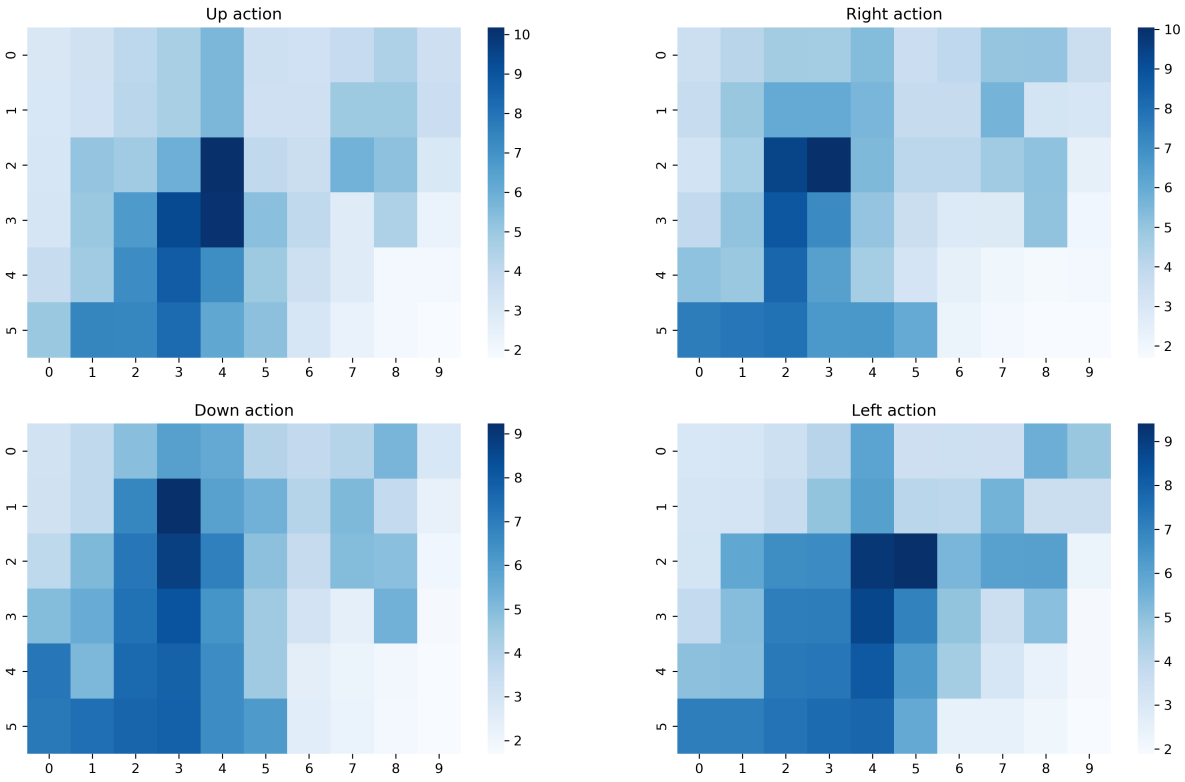


Figure 5.22: Sparse gaussian GVD. q values obtained with Average Loss QGVD after 1000 episodes with the Unified Horde.

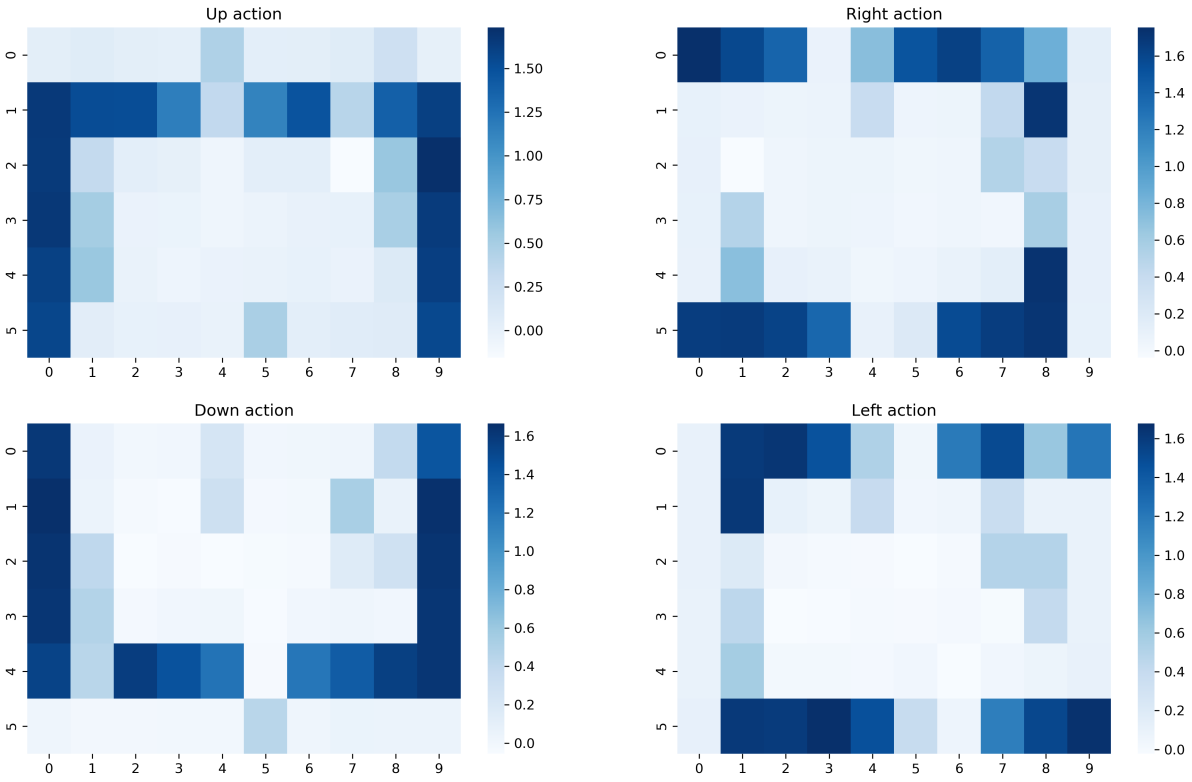


Figure 5.23: Wall detector GVD. q values obtained with Average Loss QGVD after 1000 episodes with the Unified Horde.

Chapter 6

Summary

In this Dissertation we have combined ideas from Distributional RL and General Value Function to propose the General Value Distribution framework. GVD builds on a slightly modified Distributional Bellman equation that incorporates the termination signal function γ. We have presented formally this framework and focused on QGVD, the GVD version of QDRL. We gave theoretical guarantees for GVD and QGVD, adapting the results from [2], [5], [20]. As the main appeal of GVD is to build a predictive knowledge of the horde's environment, we first focused on distributional policy evaluation. This led us to propose an Average Loss version of the One Sample evaluation method for QGVD. We experimentally showed evidence of mean and distributional convergence for both algorithms. Our findings suggest that the Average Loss should be more appropriate to evaluate a non-deterministic policy. As for control, we derived a risk-sensitive Q-Learning algorithm based on exponential utilities [13]. We experimentally obtained, in online and offline settings, the expected behaviour from the horde. We have also developed two kind of architectures, the Unified and Separate Horde, which have GVDs that respectively share and do not share a common part of their Neural Network. We showed that we could learn multiple GVDs in both cases. On the other hand, we explored connections between GVD and Kernel Mean Embedding, with the idea of learning a mean embedding of the return distribution. We proposed a naive Monte Carlo method which ignores the correlation between samples to compute a Conditional Mean Embedding estimate. We experimentally showed that we could use them to recover the q values.

6.1 Future work

We developed the Separate and Unified Horde, but did not try to compare their performances properly. We only showed that we could approximately learn the return distri-

butions with both and that using a single model did not appear to be detrimental to the overall performance. It would be interesting to see if the Unified Horde architecture actually enables a learning improvement for the main GVD when one associates multiple auxiliary GVDs. Furthermore, the code implementation of the Horde does not include parallel computation yet. The learning steps that the horde takes are done sequentially and are therefore more time consuming. Incorporating paralleling tools to speed up learning should be a good feature for a practical use of this technique.

On the other hand, many unsuccessful attempts were undertaken to derive a Temporal Difference algorithm that learns the mean embedding of the return distribution. Confronted with this difficulty, we turned to a Monte Carlo approach instead but this method is computationally costly and learning can occur only at the end of an episode. A TD method could address both issues. Furthermore, approximate matrix inversion techniques should lighten the computation costs.

Bibliography

[1] N. Aronszajn. "Theory of Reproducing Kernels". In: *Transactions of the American Mathematical Society* 68.3 (1950), pp. 337–404. ISSN: 00029947. URL: http://www.jstor.org/stable/1990404.

[2] Marc G. Bellemare, Will Dabney, and Rémi Munos. *A Distributional Perspective on Reinforcement Learning*. 2017. arXiv: 1707.06887 [cs.LG].

[3] Marc G. Bellemare et al. *Distributional reinforcement learning with linear function approximation*. 2019. arXiv: 1902.03149 [cs.LG].

[4] A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics.* Kluwer Academic Publishers, 2004.

[5] Will Dabney et al. *Distributional Reinforcement Learning with Quantile Regression*. 2017. arXiv: 1710.10044 [cs.AI].

[6] Will Dabney et al. *Implicit Quantile Networks for Distributional Reinforcement Learning*. 2018. arXiv: 1806.06923 [cs.LG].

[7] S. Grünewälder et al. "Conditional mean embeddings as regressors." In: *Proceedings of the 29th International Conference on Machine Learning (ICML)* (2012), pp. 1823–1830.

[8] S. Grünewälder et al. "Modelling transition dynamics in MDPs with RKHS embeddings." In: *Proceedings of the 29th International Conference on Machine Learning (ICML)* (2012), pp. 535–542.

[9] F. R. Bach K. Fukumizu and M. I. Jordan. "Dimensionality reduction for supervised learning with reproducing kernel Hilbert spaces." In: *Journal of Machine Learning Research* (2004), pp. 73–99.

[10] Motonobu Kanagawa and Kenji Fukumizu. "Recovering Distributions from Gaussian RKHS Embeddings". In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Proceedings of Machine Learning Research. PMLR, 2014, pp. 457–465. URL: http://proceedings.mlr.press/v33/kanagawa14.html.

[11] Guy Lever et al. "Compressed Conditional Mean Embeddings for Model-Based Reinforcement Learning". In: *AAAI*. 2016.

[12] Clare Lyle, Pablo Samuel Castro, and Marc G. Bellemare. *A Comparative Analysis of Expected and Distributional Reinforcement Learning*. 2019. arXiv: 1901.11084 [cs.LG].

[13] Chris Maddison et al. "Particle Value Functions". In: (Mar. 2017).

[14] H. Maei and R. Sutton. "GQ(lambda): A general gradient algorithm for temporal-difference prediction learning with eligibility traces". In: *AGI 2010*. 2010.

[15] Hamid Maei et al. "Toward Off-Policy Learning Control with Function Approximation". In: Aug. 2010, pp. 719–726.

[16] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[17] Y. Nishiyama et al. "Hilbert space embeddings of POMDPs." In: *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)* (2012), pp. 644–653.

[18] Skare Ø., Bølviken E., and L. Holden. "Improved Sampling-Importance Resampling and Reduced Bias Importance Sampling". In: *Scandinavian Journal of Statistics* (2003), 30: 719–737. DOI: 10.1111/1467-9469.00360.

[19] Mark Rowland et al. *An Analysis of Categorical Distributional Reinforcement Learning*. 2018. arXiv: 1802.08163 [stat.ML].

[20] Mark Rowland et al. *Statistics and Samples in Distributional Reinforcement Learning*. 2019. arXiv: 1902.08102 [stat.ML].

[21] Bernhard Schölkopf, Alexander J. Smola, and Francis Bach. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2018. ISBN: 0262536579.

[22] A. J. Smola et al. "A Hilbert space embedding for distributions". In: Springer Verlag, 2007, pp. 13–31.

[23] L. Song, K. Fukumizu, and A. Gretton. "Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models". In: *IEEE Signal Processing Magazine* (2013), pp. 98–111.

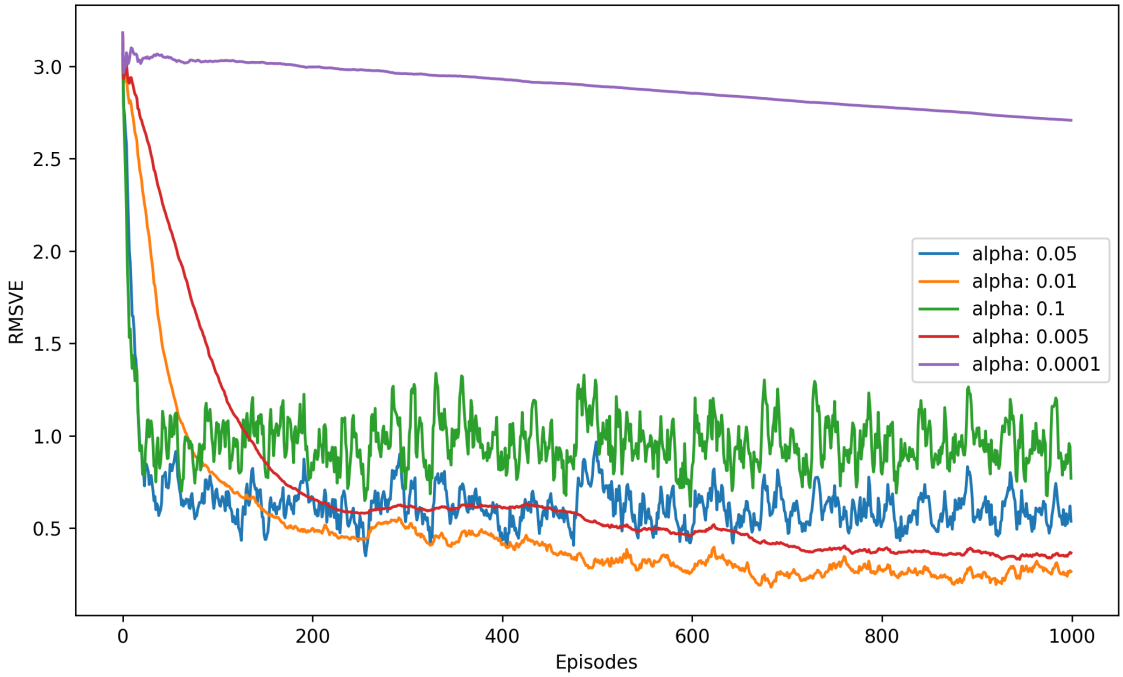
[24] L. Song et al. "Hilbert space embeddings of conditional distributions with applications to dynamical systems". In: *Proceedings of the 26th International Conference on Machine Learning (ICML)* (2009), pp. 961–968.

[25] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[26] Richard Sutton et al. “Horde : A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction Categories and Subject Descriptors”. In: vol. 2. Jan. 2011.

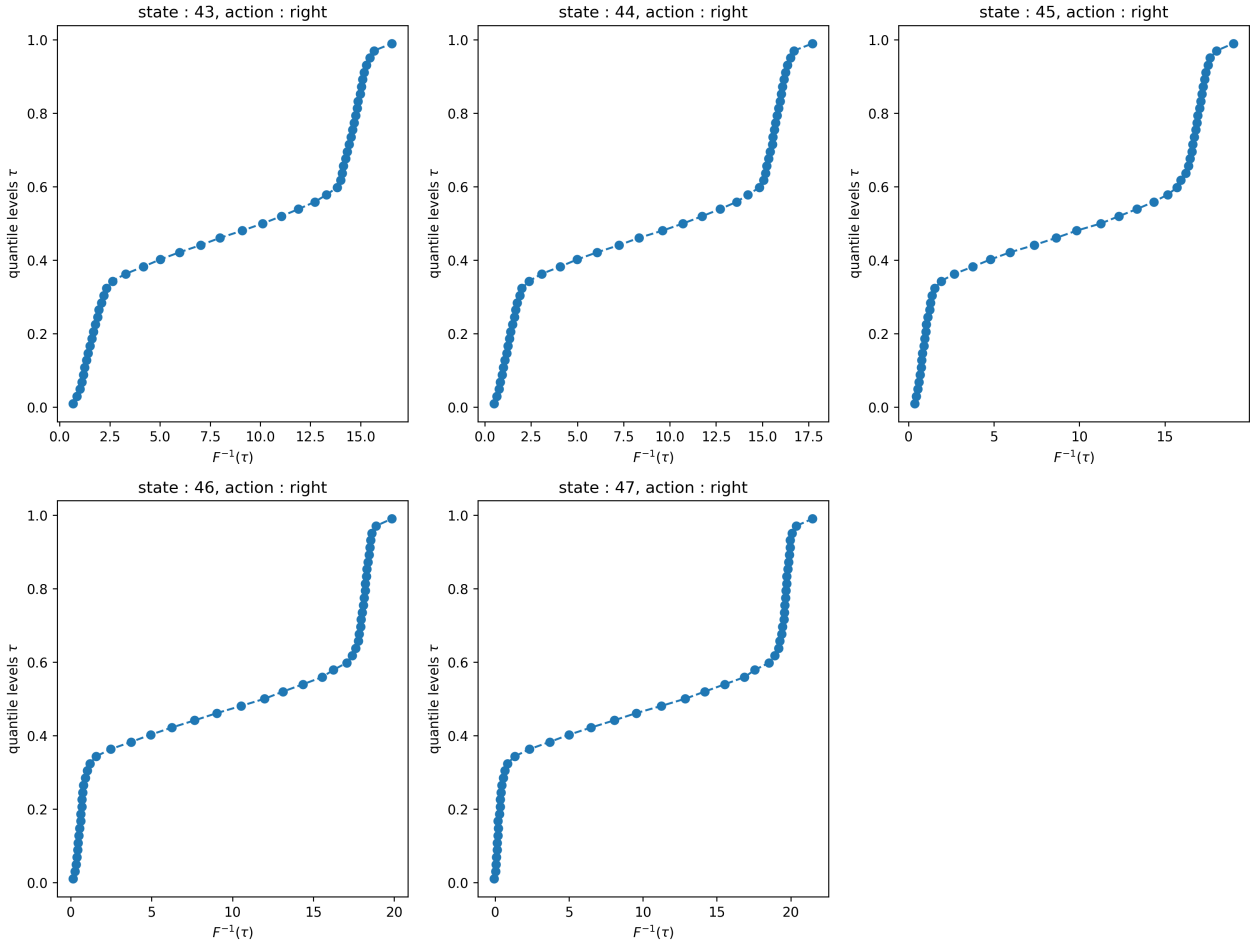
[27] Adam White et al. “Developing a predictive approach to knowledge”. In: (2015).

A - Additional Figures



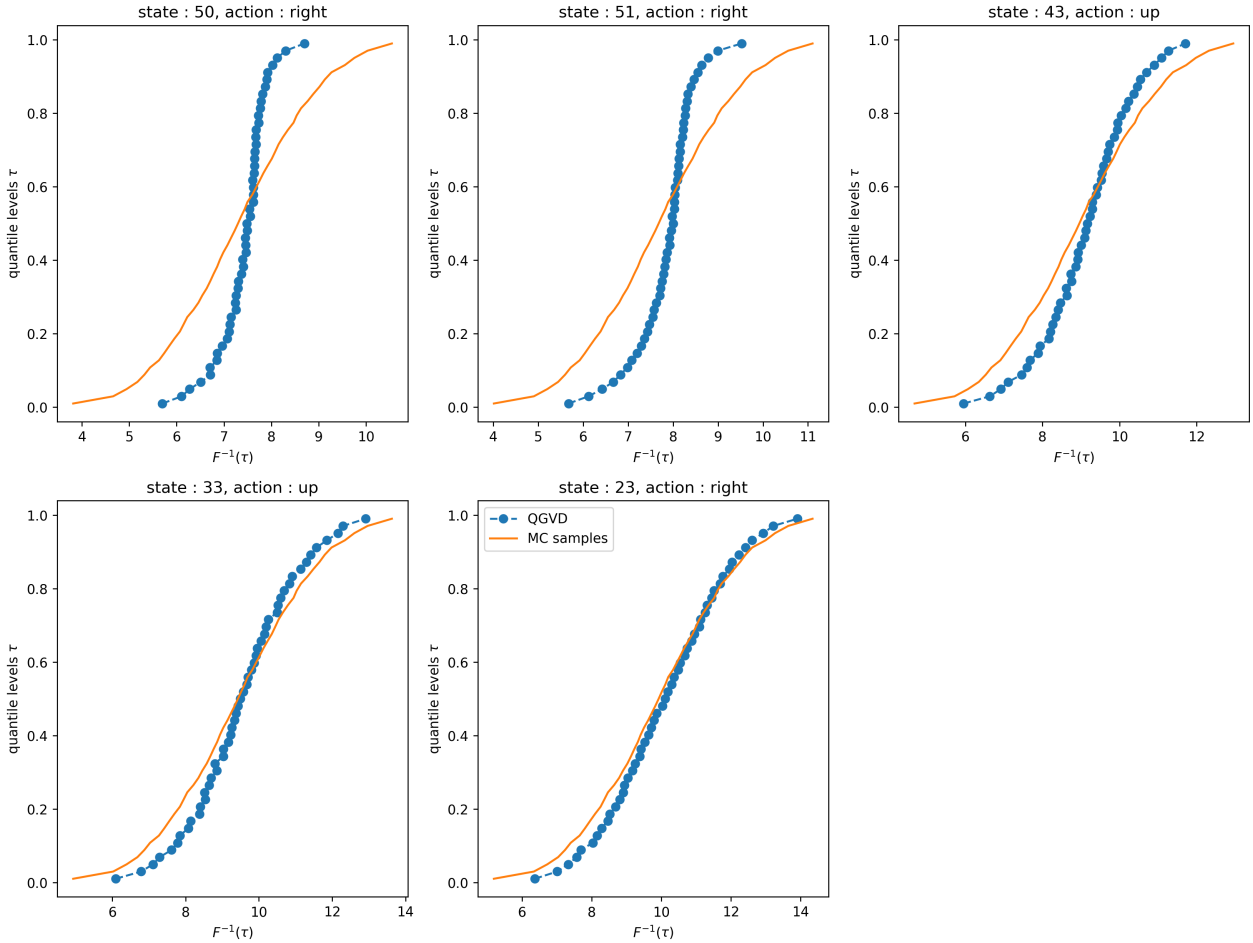
-

Figure 6.1: Average RMSVE of values obtained with GTD(λ). Same policy evaluation task as in Section 5.2.1 with π_{right}, and $c_{\text{Bernouilli}}$. We average the results over 10 runs.



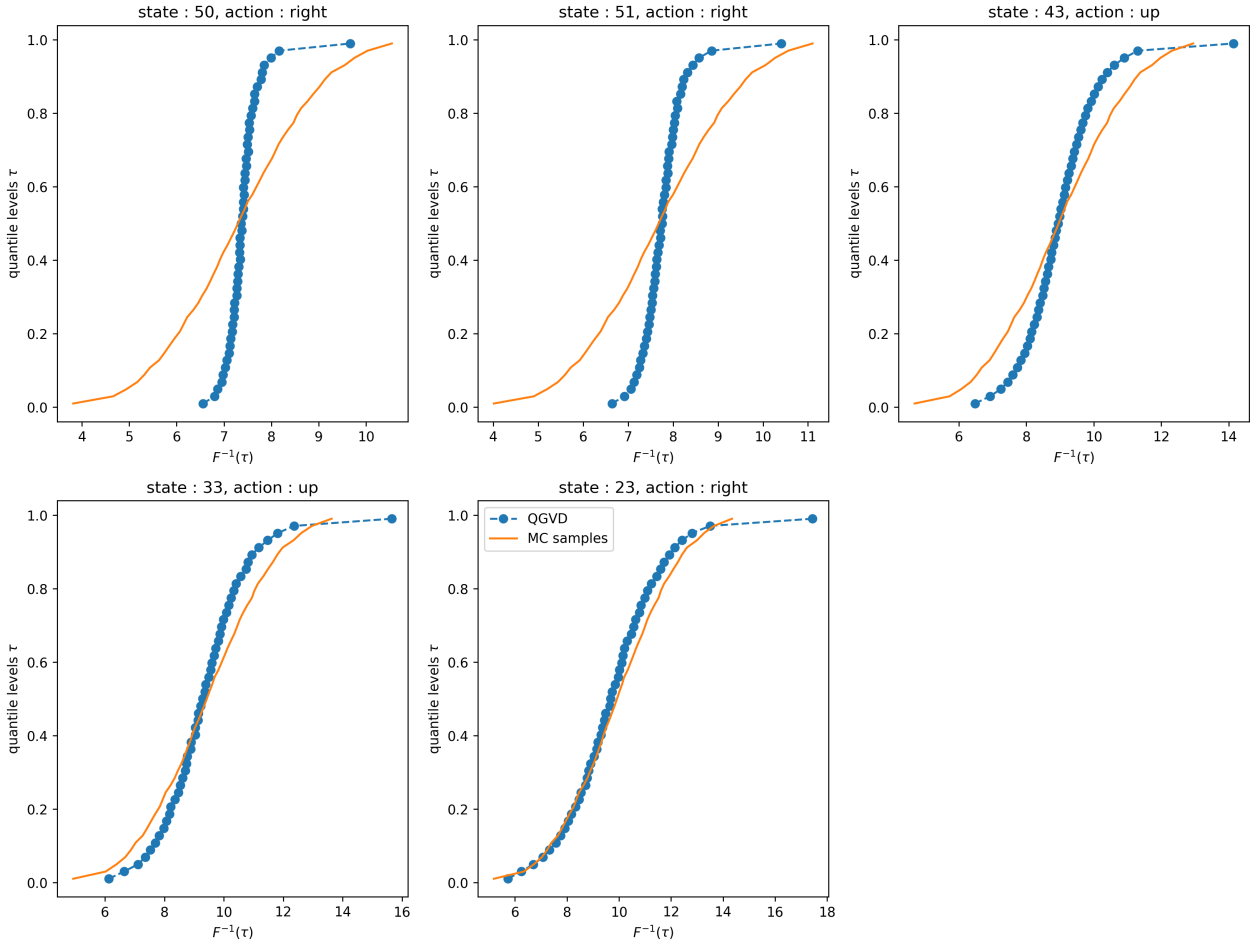
-

Figure 6.2: Quantile values obtained with Average Loss QGVD for the policy evaluation task of Section 5.2.1 with π_{right}, and $c_{\text{Bernouilli}}$. We plot the empirical cdf of the states to the left of S_G with the "right" action.



-

Figure 6.3: Quantile values obtained with Average Loss QGVD for the policy evaluation task of Section 5.3.1 with the semi-deterministic policy and c_{Gaussian}. We plot the empirical cdf of the state-action pairs of the track list.



-

Figure 6.4: Quantile values obtained with Q-Learning QGVD for the control task of Section 5.3.2 with c_{Gaussian}. We plot the empirical cdf of the state-action pairs of the track list.

B - Code

The code can be found in this github repository: github link