Neurocomputing 534 (2023) 199–219

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Distributional reinforcement learning with unconstrained monotonic neural networks

Thibaut Théate [a,*], Antoine Wehenkel [a], Adrien Bolland [a], Gilles Louppe [a], Damien Ernst [a,b]

[a] Department of Electrical Engineering and Computer Science, University of Liège, Liège, Belgium
[b] Information Processing and Communications Laboratory, Institut Polytechnique de Paris, Paris, France

ARTICLE INFO

Article history:
Received 3 June 2022
Revised 30 January 2023
Accepted 26 February 2023
Available online 3 March 2023
Communicated by Zidong Wang

Keywords:
Artificial intelligence
Machine learning
Distributional reinforcement learning
Unconstrained monotonic neural networks
Probability metrics

ABSTRACT

The distributional reinforcement learning (RL) approach advocates for representing the complete probability distribution of the random return instead of only modelling its expectation. A distributional RL algorithm may be characterised by two main components, namely the representation of the distribution together with its parameterisation and the probability metric defining the loss. The present research work considers the *unconstrained monotonic neural network* (UMNN) architecture, a universal approximator of continuous monotonic functions which is particularly well suited for modelling different representations of a distribution. This property enables the efficient decoupling of the effect of the function approximator class from that of the probability metric. The research paper firstly introduces a methodology for learning different representations of the random return distribution (PDF, CDF and QF). Secondly, a novel distributional RL algorithm named *unconstrained monotonic deep Q-network* (UMDQN) is presented. To the authors' knowledge, it is the first distributional RL method supporting the learning of *three*, *valid* and *continuous* representations of the random return distribution. Lastly, in light of this new algorithm, an empirical comparison is performed between three probability quasi-metrics, namely the Kullback–Leibler divergence, Cramer distance, and Wasserstein distance. The results highlight the main strengths and weaknesses associated with each probability metric together with an important limitation of the Wasserstein distance.

© 2023 Elsevier B.V. All rights reserved.

1. Introduction

Reinforcement learning (RL) is a family of techniques belonging to the area of *machine learning* (ML), which is concerned with the learning process of an agent sequentially interacting within an environment and aiming to maximise the notion of cumulative reward. *Deep reinforcement learning* (DRL) extends this approach by using *deep learning* (DL) techniques to generalise the information acquired from the interaction of the agent with its environment. Depending on whether a model of the environment is available and exploited or not, the RL algorithms can be either classified *model-based* or *model-free*. The present research focuses exclusively on the second category, which can be subdivided into two classes: *policy optimisation* and *Q-learning*. The RL algorithms based on the *Q-learning* approach generally model the expectation of the random return to be maximised [26]. Alternatively, the *distributional RL* approach proposes learning the entire probability distribution of the random return. This methodology presents key advantages including learning richer representations of the returns generated by the environment, which leads to more efficient and stable learning, as well as making risk-sensitive control and exploration policies possible [1,11].

A distributional RL algorithm may be characterised by two main components. The first one relates to both the representation and the parameterisation of the random return distribution. A unidimensional distribution possesses several different representations, such as its probability density function (PDF), its cumulative distribution function (CDF) and its quantile function (QF). Typically, *deep neural networks* (DNNs) are considered for approximating these various functions. The second component concerns the probability quasi-metric adopted for comparing two distributions. Multiple quasi-metrics do exist for that purpose, the main ones experimented in distributional RL being the Kullback–Leibler (KL) divergence, the Cramer distance (which is also named energy distance), and the Wasserstein distance. In the rest of this research paper, they will simply be referred to as *probability metrics*. In the context of distributional RL, the role of the probability metric is to quantitatively compare two distributions of the random return in

* Corresponding author.
E-mail addresses: thibaut.theate@uliege.be (T. Théate), antoine.wehenkel@uliege.be (A. Wehenkel), adrien.bolland@uliege.be (A. Bolland), g.louppe@uliege.be (G. Louppe), dernst@uliege.be (D. Ernst).

https://doi.org/10.1016/j.neucom.2023.02.049
0925-2312/© 2023 Elsevier B.V. All rights reserved.

order to apply a *temporal difference* (TD) learning method, in a similar way to the mean squared error between Q-values in classical RL. The choice of the probability metric is particularly important since each metric offers different theoretical convergence guarantees for distributional RL.

The core idea of this research work is to consider the *unconstrained monotonic neural network* (UMNN) architecture [27] in the scope of distributional RL. Originally designed for autoregressive flows, this particular architecture is in fact a universal approximator of continuous monotonic functions. Several works have already demonstrated the ability of this neural network to accurately model continuous monotonic functions in practice [17,24]. Since both the CDF and QF are monotonic, the UMNN architecture is expected to offer superior capability compared to classical neural networks when it comes to representing distributions. Moreover, the PDF can also be efficiently represented by this architecture, when standing at the heart of a *normalizing flow* [18] by taking advantage on the *change of variables theorem* [27]. Because the single UMNN architecture can effectively model different representations of the random return distribution, it enables the efficient decoupling of the effect of the function approximator class from that of the probability metric, making a fair comparison between probability metrics possible.

This leads to the main contributions of the present research work, which are threefold. Firstly, the paper introduces a methodology for learning three representations of the random return probability distribution, namely the PDF, CDF and QF. Secondly, the article presents a novel distributional RL algorithm, denominated *unconstrained monotonic deep Q-network* (UMDQN), combining a UMNN with this new methodology for learning different valid representations of the continuous distribution of the random return. Thirdly, taking advantage of this innovative algorithm, the research work proposes an empirical comparison of three probability metrics commonly used in distributional RL, namely the KL divergence, the Cramer distance and the Wasserstein distance. This analysis highlights the main strengths and weaknesses associated with each probability metric, but also reveals an important limitation of the Wasserstein distance. Actually, the observed limitation highlights a critical approximation made by several state-of-the-art distributional RL algorithms, leading to the learning of inaccurate distributions for the random return. To the authors' knowledge, the proposed algorithmic solution is the first distributional RL approach supporting the learning of *several* (PDF, CDF and QF), *valid* (by ensuring monotonicity) and *continuous* (as opposed to discrete) representations of the random return distribution. To end this introductory section, it should be emphasised that the core objective of this research work is not to present a novel distributional RL algorithm competing with the state-of-the-art algorithms on a given testbench, typically the Atari-57 benchmark [2], but rather to empirically derive new insights about distributional RL from this algorithmic solution.

The present research paper is structured as follows. First of all, a concise review of the scientific literature about distributional RL in general and the state-of-the-art algorithms is presented in Section 2. Afterwards, Section 3 formally introduces the distributional RL approach together with the mathematical notations adopted in this research work. Then, Section 4 presents in detail the novel distributional RL algorithm proposed, together with a methodology for learning different representations of the random return probability distribution. Following on, Section 5 presents the performance assessment methodology adopted and discusses the results achieved by this new distributional RL algorithm. Finally, Section 6 draws some conclusions and briefly discusses interesting leads for future work.

2. Literature review

Q-learning is a *model-free* RL approach based on the learning of the quantity Q representing the quality of executing a certain action in a particular state [26]. Originally based on tabular or linear approximations, the DQN algorithm [13] extends this approach by using a DNN for approximating the quantity Q in a non-linear setting. New to the field is the distributional RL approach advocating for learning the entire probability distribution of the random return instead of only modelling its expectation [1]. Fundamental research on distributional RL is still in its early stages, but key benefits have already been discovered [11,20].

Several distributional RL algorithms can be found in scientific literature, based on diverse representations of the random return distribution but also different probability metrics. The *categorical DQN* (CDQN) algorithm [1], also known as *C51*, approximates the PDF of the random return through categorical distributions and uses the KL divergence for quantitatively comparing these distributions. The link between this original distributional RL algorithm and the Cramer distance probability metric was later highlighted [19]. Alternatively, the *quantile regression DQN* (QR-DQN) algorithm [7] learns the distribution of the random return by manipulating the QF with fixed uniform quantile values and the Wasserstein distance. Compared to the CDQN algorithm, this approach has the key advantage of avoiding the specification of a fixed support for the random return values. Nevertheless, both algorithms suffer from the same drawback of estimating the distribution of the random return on fixed locations (either value or probability), with as a consequence that the distributions learnt are discrete. The *implicit quantile network* (IQN) algorithm [6] solves this problem by learning the quantile values from quantile fractions sampled from a uniform distribution $\mathcal{U}([0,1])$. This is achieved with a specific DNN representing the QF by mapping quantile fractions to quantile values and trained by minimising the Wasserstein distance. Finally, the *fully parameterised quantile function* (FQF) algorithm [28] extends the previous methodology by parameterising both quantile fraction and value axes. To do so, two DNNs are used: one for generating appropriate quantile fractions and one for mapping these quantile fractions to quantile values. They are jointly trained by minimising the Wasserstein distance once again. Fig. 1 illustrates these distributional RL algorithms in the context of the Atari-57 benchmark [2].

Besides the previous distributional RL algorithms that are well-established in the research community, one can mention several recent research works bringing new interesting insights about distributional RL. The *moment matching DQN* (MMDQN) algorithm [15] learns via a DNN a finite set of statistics for the distribution of the random return by implicitly matching all orders of moments between the random return distribution and its target. The key benefit of this approach is to avoid the predefined statistic principle used in prior distributional RL works, which leads to a simpler objective amenable to backpropagation. This is achieved by learning unrestricted statistics, i.e. deterministic samples, of the random return distribution by leveraging the maximum mean discrepancy technique from hypothesis testing. Sharing a similar philosophy to the present research paper, the *non-crossing QR-DQN* algorithm [32] is an improvement of the well-established QR-DQN algorithm implementing non-crossing quantile regression to ensure the monotonicity constraint for the QF. This enhancement is built on the observation that the non-decreasing property of learnt quantile curves is not guaranteed, which leads to abnormal distribution estimates and reduced model interpretability. However, this technique is not directly transferable to the IQN or FQF algorithms. To end this literature review, an important study reveals that the reward system of the human brain would operate similarly to

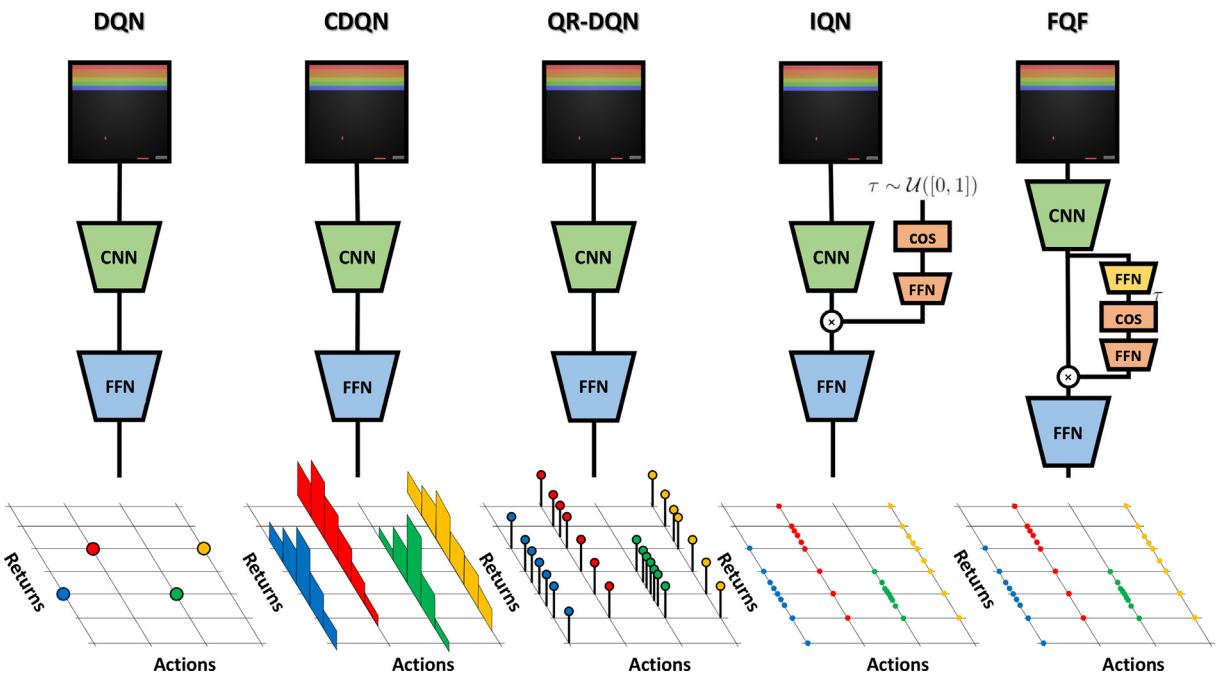


Fig. 1. Main distributional RL algorithms from scientific literature, for the Atari-57 benchmark [2].

distributional RL [5]. Indeed, the findings suggest that the human brain represents possible future rewards as a complete probability distribution and not as a single mean of stochastic outcomes. This is naturally very encouraging news supporting the soundness of the distributional RL approach.

3. Distributional reinforcement learning

This research paper adopts the standard RL setting where the agent interacts with its environment modelled as a *Markov decision process* (MDP). An MDP is a 6-tuple $(\mathcal{S},\ \mathcal{A},\ p_R,\ p_T,\ p_0,\ \gamma)$ where $\mathcal{S}$ and $\mathcal{A}$ respectively are the state and action spaces, $p_R(r|s,a)$ is the probability distribution from which the reward $r \in \mathbb{R}$ is drawn given a state-action pair (s,a), $p_T(s'|s,a)$ is the transition probability distribution, $p_0(s_0)$ is the probability distribution over the initial states $s_0 \in \mathcal{S}$, and $\gamma \in [0,1[$ is the discount factor. The RL agent makes decisions according to its policy $\pi : \mathcal{S} \to \mathcal{A}$, which is considered deterministic in the rest of this research paper, mapping the states $s \in \mathcal{S}$ to the actions $a \in \mathcal{A}$ (See Table 1).

The Q-learning approach focuses on modelling the *state-action value function* Q^π of a policy π. This quantity represents the expected discounted sum of rewards to be obtained by executing an action a in a state s and then following a policy π, and satisfies the *Bellman equation* [3]:

$$Q^\pi(s,a) = \mathbb{E}_{s_t,r_t}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right], \qquad (s_0,a_0) := (s,a),\ a_t = \pi(s_t)\ , \tag{1}$$

$$Q^\pi(s,a) = \mathbb{E}_{s',r}\left[r + \gamma Q^\pi(s',\pi(s'))\right]. \tag{2}$$

In a similar way, the *optimal policy* π^* based on the *optimal state-action value function* Q^* can be defined as the following:

$$Q^*(s,a) = \mathbb{E}_{s',r}\left[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')\right], \tag{3}$$

$$\pi^*(s) \in \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s,a)\ . \tag{4}$$

Distributional RL aims at modelling the entire probability distribution over returns instead of only its expectation. To this end, let the reward $R(s,a)$ be a random variable distributed under $p_R(\cdot|s,a)$, the *state-action value distribution* $Z^\pi \in \mathcal{Z}$ of a policy π is a random variable defined as follows:

$$Z^\pi(s,a) \overset{D}{=} \sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\ ,$$
$$(s_0,a_0) := (s,a),\ a_t = \pi(s_t),\ s_{t+1} \sim p_T(\cdot|s_t,a_t)\ , \tag{5}$$

where $A \overset{D}{=} B$ denotes the equality in distribution between the random variables A and B. Therefore, the state-action value function Q^π is the expectation of the *random return* Z^π. In the same way, there is a *distributional Bellman equation* recursively describing Z^π:

$$Z^\pi(s,a) \overset{D}{=} R(s,a) + \gamma P^\pi Z^\pi(s,a)\ , \tag{6}$$

$$P^\pi Z^\pi(s,a) :\overset{D}{=} Z^\pi(s',a')\ , \qquad s' \sim p_T(\cdot|s,a),\ a' = \pi(s')\ , \tag{7}$$

where $P^\pi : \mathcal{Z} \to \mathcal{Z}$ is the transition operator. Finally, one can define the *distributional Bellman operator* $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ and the *distributional Bellman optimality operator* $\mathcal{T}^* : \mathcal{Z} \to \mathcal{Z}$ as follows:

Table 1
Key characteristics (representation of the random return probability distribution and probability metric) of the main state-of-the-art distributional RL algorithms.

Algorithm	Probability distribution representation	Probability metric
DQN	Expectation (non distributional RL)	L1 metric
CDQN	Categorical PDF (fixed discrete support)	KL divergence
QR-DQN	Discrete QF (fixed quantile fractions)	Wasserstein distance
IQN	Continuous QF (quantiles drawn from $\mathcal{U}([0,1])$)	Wasserstein distance
FQF	Continuous QF (quantiles sampled by a DNN)	Wasserstein distance

$$\mathcal{T}^{\pi} Z^{\pi}(s,a) \overset{D}{=} R(s,a) + \gamma P^{\pi} Z^{\pi}(s,a) \ , \tag{8}$$

$$\mathcal{T}^{*} Z^{*}(s,a) \overset{D}{=} R(s,a) + \gamma Z^{*}(s', \pi^{*}(s')), \qquad s' \sim p_T(\cdot|s,a) \ . \tag{9}$$

Theoretically, the distributional Bellman operator $\mathcal{T}^{\pi}$ may potentially be a contraction mapping or not depending on the probability metric. This property implies that there exists a unique fixed point Z^{π} to converge towards when repeatedly applying the operator $\mathcal{T}^{\pi}$. For the distributional Bellman optimality operator $\mathcal{T}^{*}$, another condition is required for this contraction mapping property to hold: the optimal policy π^{*} has to be unique [1]. Multiple probability metrics do exist for quantitatively comparing the probability distributions of two continuous random variables. In this research work, the emphasis is set on the three main probability metrics used in distributional RL, namely the KL divergence, Cramer distance and Wasserstein distance. Table 2 formally introduces these probability metrics, together with their impact on the contraction mapping property of the distributional Bellman operator $\mathcal{T}^{\pi}$.

4. Unconstrained monotonic deep Q-network

4.1. Learning different representations of a probability distribution

This section presents a methodology for learning different representations of the probability distribution of the random return: the PDF, CDF and QF. The learning process is based on the comparison of the left- and right-hand sides of the distributional Bellman Eq. (6). For a given probability metric $\mathcal{L}$, the random return Z^{π} is a fixed point of the Bellman operator $\mathcal{T}^{\pi}$ if it minimises the following loss:

$$\mathcal{L}(\mathcal{T}^{\pi} Z^{\pi}(s,a),\ Z^{\pi}(s,a)). \tag{10}$$

for all state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$. The distributional RL problem at hand will be addressed by defining a hypothesis space for the quantity Z^{π} and minimising the loss function (10) over this space using *stochastic gradient descent* (SGD). In the following, the effect of the distributional Bellman operator $\mathcal{T}^{\pi}$ on the different representations of the random return distribution is rigorously studied. Intuitively, the discount factor γ *squeezes* the random return distribution while the reward R *shifts* this probability distribution, as illustrated in Fig. 2 in the simplified situation of deterministic reward and transition function.

PDF representation Let $p_{Z^{\pi}}(z|s,a)$ be the PDF of the random variable Z^{π} given the state-action pair (s,a) at the return z. Assuming the KL divergence $\mathcal{L}_{KL}$ as the probability metric considered, the loss to be minimised defined in Eq. (10) can be re-expressed as follows:

$$\mathcal{L}_{KL}(\mathcal{T}^{\pi} Z^{\pi}(s,a),\ Z^{\pi}(s,a)) = D_{KL}(p_{\mathcal{T}^{\pi} Z^{\pi}}(z|s,a),\ p_{Z^{\pi}}(z|s,a)) \tag{11}$$

$$= D_{KL}\left(\mathbb{E}_{s',r}\left[\frac{1}{\gamma}\ p_{Z^{\pi}}\left(\frac{z-r}{\gamma}|s',\pi(s')\right)\right], p_{Z^{\pi}}(z|s,a)\right). \tag{12}$$

CDF representation Let $F_{Z^{\pi}}(z|s,a)$ be the CDF of the random variable Z^{π} conditioned by the state-action pair (s,a) at the return z. Assuming the Cramer distance $\mathcal{L}_C$ as the probability metric considered, the loss formally defined in Eq. (10) can be re-expressed as follows:

$$\mathcal{L}_C(\mathcal{T}^{\pi} Z^{\pi}(s,a),\ Z^{\pi}(s,a)) = D_C(F_{\mathcal{T}^{\pi} Z^{\pi}}(z|s,a),\ F_{Z^{\pi}}(z|s,a)) \tag{13}$$

$$= D_C\left(\mathbb{E}_{s',r}\left[F_{Z^{\pi}}\left(\frac{z-r}{\gamma}|s',\pi(s')\right)\right], F_{Z^{\pi}}(z|s,a)\right). \tag{14}$$

QF representation Let $F^{-1}_{Z^{\pi}}(\tau|s,a)$ be the QF of the random variable Z^{π} given the state-action pair (s,a) at the quantile fraction $\tau \in [0,1]$. Assuming the Wasserstein distance $\mathcal{L}_W$ as the probability metric considered, the loss to be minimised defined in Eq. (10) can be re-expressed as follows:

$$\mathcal{L}_W(\mathcal{T}^{\pi} Z^{\pi}(s,a),\ Z^{\pi}(s,a)) = D_W\left(F^{-1}_{\mathcal{T}^{\pi} Z^{\pi}}(\tau|s,a),\ F^{-1}_{Z^{\pi}}(\tau|s,a)\right) \tag{15}$$

$$\simeq D_W\left(\mathbb{E}_{s',r}\left[r + \gamma F^{-1}_{Z^{\pi}}(\tau|s',\pi(s'))\right], F^{-1}_{Z^{\pi}}(\tau|s,a)\right). \tag{16}$$

As far as mathematical proofs are concerned, Eqs. (12) and (14) are respectively supported by Proposition 1 and Corollary 1 in Appendix A. On the contrary, Eq. (16) could not be rigorously proven as originally intended. In order to get a better understanding of the challenge faced, some basic experiments have been conducted. The results suggest that Eq. (16) results from an approximation of $F^{-1}_{\mathcal{T}^{\pi} Z^{\pi}}(\tau|s,a)$, leading to a random variable with the correct expectation but potentially different higher-order moments. In the scope of distributional RL, such an approximation may have two completely different implications depending on the objective pursued. If the intention is to accurately learn the probability distribution of the random return for implementing risk-aware policies, this approximation is obviously problematic. On the contrary, if the goal is to learn policies maximising the expectation of the random return, this approximation may have no negative effect since the distribution learnt has the correct first-order moment. In fact, this approach is adopted by the state-of-the-art QR-DQN, IQN and FQF algorithms which are able to learn valuable policies in practice, based on the expectation of the random return alone [7,6,28].

4.2. Unconstrained monotonic neural network

The PDF, CDF and QF of continuous random variables share the important property of being effectively modelled with strictly monotonic functions. This is the main reason for this research paper to consider *unconstrained monotonic neural networks* (UMNNs), which are universal approximators of continuous monotonic functions, for parameterising the random return probability distribution. Formally, a UMNN defines a parametric continuous monotonic function $G(\cdot;\theta) : \mathbb{R} \to \mathbb{R}$ as follows:

$$G(x;\theta) := \int_0^x g(t;\theta)dt + \beta \ , \tag{17}$$

where $g(\cdot;\theta) : \mathbb{R} \to \mathbb{R}^+$ is a free-form neural network whose output positiveness is enforced via an appropriate activation function (e.g. ReLU or exponential), where θ denotes its parameters, and

Table 2
Formal definition of the probability metrics studied, where A and B are two random variables, and where p_D, F_D and F_D^{-1} denote the PDF, CDF and QF of the random variable D, respectively.

Probability metric		$\mathcal{T}^{\pi}$ contraction?
KL divergence	$\mathcal{L}_{KL}(A,B) = D_{KL}(p_A, p_B) = \int_{\mathbb{R}} p_A(x) \log\left(\frac{p_A(x)}{p_B(x)}\right)dx$	No [14]
Cramer distance	$\mathcal{L}_C(A,B) = D_C(F_A, F_B) = \left(\int_{\mathbb{R}} (F_A(x) - F_B(x))^2 dx\right)^{1/2}$	Yes [19]
Wasserstein distance	$\mathcal{L}_W(A,B) = D_W(F_A^{-1}, F_B^{-1}) = \int_0^1 F_A^{-1}(x) - F_B^{-1}(x)dx$	Yes [1]

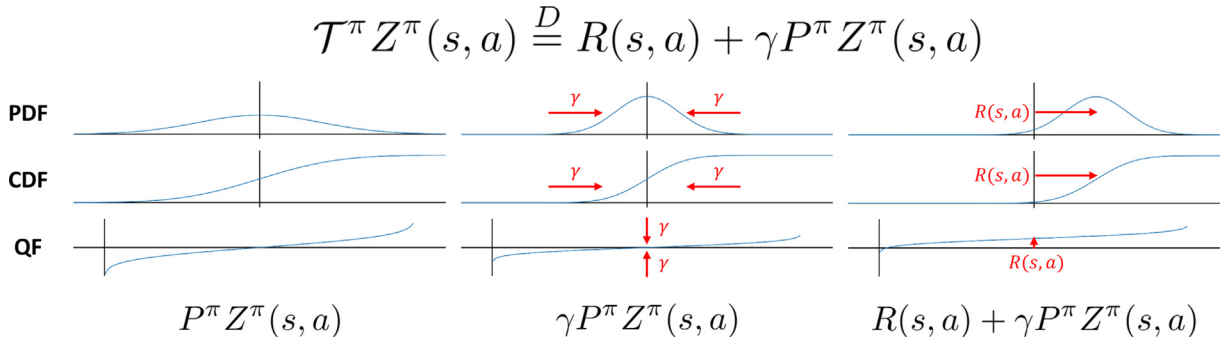


Fig. 2. Illustration of the effect of the distributional Bellman operator on different representations of the random return probability distribution in the simplified situation of both a deterministic reward and a deterministic transition function.

where $\beta \in \mathbb{R}$ is a trainable scalar parameter. This parameterisation can efficiently generalise to random variables conditioned by other quantities, e.g., the state s and the action a. A natural solution is to add these conditioning variables c as an additional vector input to the neural network g and to parameterise β as another neural network. In this particular case, Eq. (17) can be re-expressed as follows:

$$G(x|c;\theta) := \int_0^x g(t,c;\theta_g)dt + \beta(c;\theta_\beta) \ , \tag{18}$$

where the parameters of the monotonic transformation are $\theta = \theta_g \cup \theta_\beta$. Evaluating the function G requires solving an integral, which is performed numerically via Clenshaw-Curtis quadrature.

In the scope of distributional RL, the QF of the random return Z taking as inputs quantile fractions $\tau \in [0,1]$ can be parameterised by a UMNN as $F_Z^{-1}(\tau|s,a;\theta) := G(\tau|s,a;\theta)$. Modelling the CDF of the random return Z requires the output to be bounded in $[0,1]$, which is achieved by passing the output of the UMNN through a sigmoid function $\sigma : F_Z(z|s,a;\theta) := \sigma(G(z|s,a;\theta))$. Modelling the random return PDF $p_Z(z|s,a;\theta)$ can be done via *normalizing flows* [18]. More precisely, it is achieved by using a fixed latent distribution p_Y and exploiting the property that there exists a unique continuous monotonic function f satisfying the following equation (*change of variables theorem*) [27]:

$$p_Z(z|s,a;\theta) = p_Y(f(z|s,a;\theta))\left|\frac{\partial f}{\partial z}\right| \ . \tag{19}$$

The representation of p_Z is achieved by modelling the function f with a UMNN and fixing p_Y to an isotropic normal distribution. With such a representation, drawing samples from p_Z is performed by drawing samples from p_Y and applying the function f^{-1}. This requires inverting the UMNN, which can be done numerically by using any inversion method such as a binary search, since the inverse of a monotonic function is also monotonic. Appendix B provides additional information about the use of UMNNs in this research work.

4.3. Unconstrained monotonic deep Q-network algorithm

This section presents the *unconstrained monotonic deep Q-network* (UMDQN) algorithm, a novel generic distributional RL algorithm based on the methodology introduced in Section 4.1 and working with the UMNN architecture for validly representing the continuous probability distribution of the random return. More precisely, this research work details three versions of the generic UMDQN distributional RL algorithm: the UMDQN-KL, UMDQN-C and UMDQN-W algorithms, which respectively approximate the continuous PDF, CDF and QF of the random return Z^π by minimis-

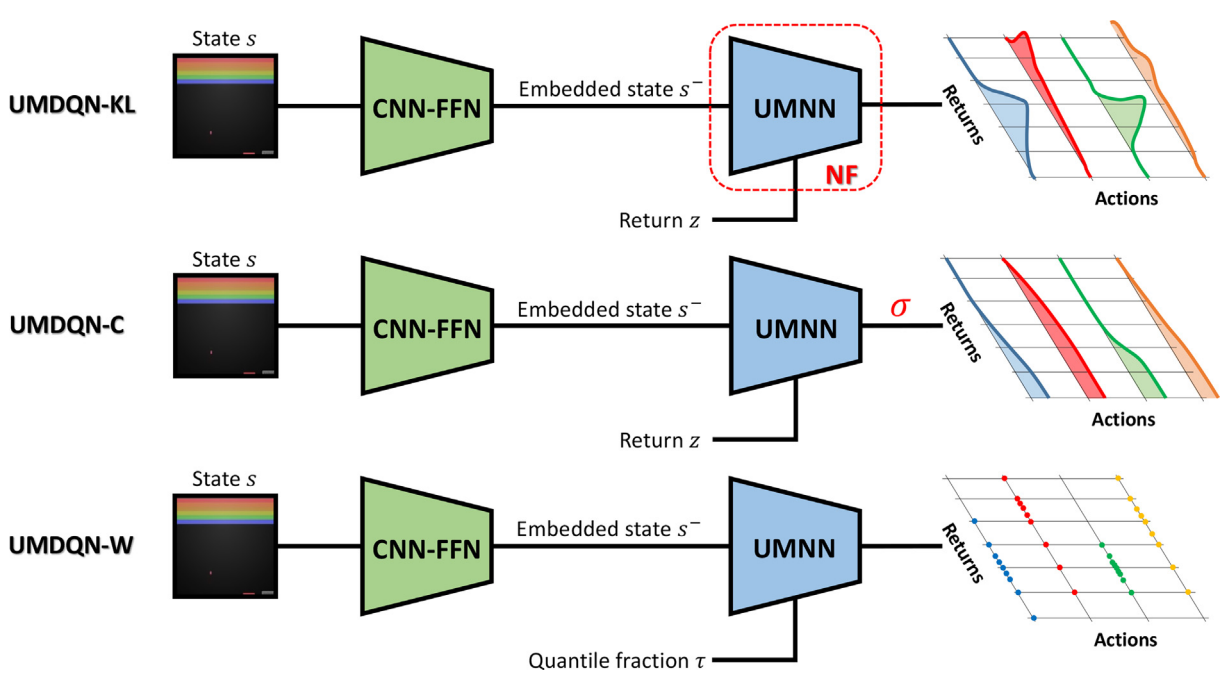


Fig. 3. Illustration of the three versions of the UMDQN algorithm in the context of Atari games.

Algorithm 1: Learning process of the UMDQN algorithm

Sample a batch of N_e experiences $e = (s, a, r, s')$ from the replay memory.
Determine for each experience the next optimal action $a' = \pi(s') = \text{argmax}_{a \in \mathcal{A}} \mathbb{E}[G_Z(s', a; \theta^-)]$.
Compute the loss $\hat{\mathcal{L}} = \frac{1}{N_e} \sum_{s,a,r,s'} \left[\sum_{x \in \mathcal{X}} \left[L\left(T^\pi G_Z(x|s,a;\theta^-),\ G_Z(x|s,a;\theta)\right) \right] \right]$.
Optimise the UMNN parameters θ according to the resulting gradients $\nabla \hat{\mathcal{L}}$.

ing the KL divergence, Cramer distance and Wasserstein distance. Therefore, in contrast to previous works on distributional RL, the proposed approach presents the key advantage of offering a choice regarding the representation of the probability distribution together with the probability metric to work with. An illustration of the three novel distributional RL algorithms in the context of Atari games is provided in Fig. 3.

The UMDQN algorithm is an *off-policy* and *value iteration* DRL algorithm which is based on the same procedure as the DQN algorithm for generating trajectories and learning from that information. Numerous experiences $e = (s, a, r, s')$ are generated by sequentially interacting with the environment and are stored into an *experience replay memory* of fixed size with a first-in-first-out (FIFO) replacement policy. Additionally, a *target network*, whose parameters are denoted θ^-, is used for fixing the Bellman probability distribution to be learnt and is updated at regular intervals. As far as exploration is concerned, it is ensured through the use of the ϵ-greedy technique. At regular intervals during the interactions between the agent and its environment, batches of experiences are sampled from the replay memory to compute *Monte Carlo* (MC) estimates of an approximation of the loss defined in Eq. (10) and perform stochastic gradient descent.

In fact, three important approximations are made regarding the loss defined in Eq. (10). The first one results from the evaluation of the loss in expectation over the distribution of state-action pairs *sampled* from the environment. The second approximation originates from the fact that the expectation $\mathbb{E}_{s',r}$ in Eqs. (12), (14), (16) is computed *outside* the probability metric $\mathcal{L}$. The last approximation comes from the estimation of the two expectations $\mathbb{E}_{s,a}$ and $\mathbb{E}_{s',r}$ using *Monte Carlo* with the experiences sampled from the replay memory. The second approximation may potentially introduce a bias, as it has already been demonstrated for the Wasserstein distance [1]. However, there is a solution for this probability metric in particular: the *(conditional) quantile regression* method [10]. This approach is claimed to allow for the unbiased stochastic approximation of the QF, and is adopted in the QR-DQN, IQN and FQF algorithms.

The learning process of the UMDQN algorithm is described in Algorithm 1. Within this description, $G_Z(\cdot|s,a;\theta)$ denotes the random return probability distribution modelled by a UMNN with parameters θ for the state-action pair (s, a), the operator T^π is defined in Eq. (20) and reproduces the effect of the distributional Bellman operator on $G_Z(\cdot|s,a;\theta)$ in line with Eqs. (12), (14) and (16), the function L computes the error according to the probability metric selected, and $\mathcal{X}$ is a discretisation of the domain of the function representing probability distribution of the random return (PDF, CDF or QF). In this research work, the policy π considered simply selects the action maximising the expectation of the random return Z^π learnt so far. The detailed pseudocodes of the three versions of the UMDQN algorithm, together with some implementation details, are provided in Appendix C.

$$T^\pi G_Z(x|s,a;\theta) = \begin{cases} \frac{1}{\gamma} G_Z\left(\frac{x-r}{\gamma}|s', \pi(s');\theta\right) & \text{if the UMNN models a PDF,} \\ G_Z\left(\frac{x-r}{\gamma}|s', \pi(s');\theta\right) & \text{if the UMNN models a CDF,} \\ r + \gamma G_Z(x|s', \pi(s');\theta) & \text{if the UMNN models a QF.} \end{cases} \tag{20}$$

5. Results

5.1. Benchmark environments

The performance assessment methodology adopted by this research work to evaluate the performance of the UMDQN distributional RL algorithm includes four different types of benchmark environments:

- a stochastic grid world environment,
- a set of classic control environments,
- a set of Atari games,
- a set of MinAtar games.

The first benchmark environment is a *stochastic grid world* designed in the scope of this particular research on distributional RL. It consists of a 7×7 grid world within which an agent has to reach a fixed target location while avoiding a fixed trap. In order to provide sound and interesting analyses in relation to the distributional RL approach, both transition and reward functions are set stochastic (p_T and p_R). In addition to evaluating the policy performance, this specific environment will be particularly useful for visualising and interpreting the random return probability distributions learnt by the distributional RL algorithm.

The next type of benchmark environment is a set of *four classic control* problems from OpenAI Gym [4]: CartPole, Acrobot, MountainCar and LunarLander. Although the distributional RL community generally prefers Atari games to these simpler environments, they remain particularly valuable and popular benchmarks for evaluating RL algorithms. Moreover, these environments are promoted by the article [16] which proposes an alternative set of benchmarks which are less computationally intensive. That particular work being interesting and well received by the RL research community, this research paper adopts its suggestions.

The third type of benchmark environment is a set of *three Atari games* from the Atari-57 benchmark [2]: Pong, Boxing and Freeway. Distributional RL algorithms are generally evaluated on the complete Atari-57 benchmark, which offers a relevant performance assessment methodology but also presents some drawbacks for distributional RL. Indeed, the environments are mostly deterministic and require a tremendous amount of computational power. Since the original publication of the Atari-57 benchmark, diverse evaluation methodologies have progressively appeared. In this research work, the best practices proposed by the article [12] are adopted. Moreover, the mostly deterministic transitions within Atari games are made stochastic by using the sticky action generalisation technique. This last addition makes the Atari environments from the present work slightly more complex compared to the ones from previous publications in distributional RL.

The last type of benchmark environment is a set of *five MinAtar games* [29]: Asterix, Breakout, Freeway, Seaquest and SpaceInvaders. These environments are miniaturised and slightly simplified versions of several Atari games representative of the complete Atari-57 suite. The core objective behind these MinAtar environments is to make RL experimentation around Atari games more accessible and efficient. Moreover, this alternative benchmark is also promoted by the same article [16] as a replacement for the

Atari-57 benchmark in order to achieve more inclusive DRL research.

More information about these benchmark environments is provided in Appendix D. To end this section about the performance assessment methodology, an argument for bypassing the complete Atari-57 benchmark generally adopted in research works about distributional RL is presented. As previously hinted, the computational cost associated with this particular benchmark is significant. In this case, two entire weeks' worth of computations are required for training one RL agent on a single Atari game using the UMDQN algorithm with hardware acceleration enabled (NVIDIA RTX 2080 Ti). Therefore, running this novel distributional RL algorithm for five different random seeds on the complete Atari-57 benchmark would approximately require $57 * 5 * 14 \simeq 4000$ days when having access to a single GPU, without even considering the hyperparameters tuning phase. It naturally becomes totally impracticable without parallelisation with numerous GPUs. Although the UMDQN algorithm presents the drawback of being slightly more computationally expensive compared to the state-of-the-art distributional RL algorithms, the previous conclusion remains in line with findings from the scientific literature. For instance, the simpler DQN algorithm takes roughly 1425 days to fully train for each Atari game using specialised hardware (NVIDIA Tesla P100) [16]. Because this problem creates a real barrier to entry for modest laboratories having access to a limited amount of computational power, it is repeatedly discussed by the RL research community. For this reason, the present research work adopts a different yet insightful set of benchmark environments for evaluating distributional RL algorithms, based on the article [16] for more inclusive DRL research.

5.2. Results discussion

Random return distribution visualisation Besides the evaluation of the resulting policy performance, it is important to assess the correctness of the probability distributions learnt by a distributional RL algorithm. As previously hinted, this particular analysis is performed on the stochastic grid world environment. Since the underlying control problem is relatively easy to solve from a human perspective, an optimal policy can be manually derived for that specific environment. That property significantly eases the assessment of the soundness of the random return probability distributions learnt. Once the optimal policy is available, the true probability distributions of the random return can be effectively estimated via Monte Carlo. The same operation should also be performed with the policy learnt by the distributional RL algorithm, since incorrect probability distributions could also be caused by suboptimal policies. Based on that methodology, Fig. 4 graphically compares the probability distributions learnt by the three versions of the UMDQN algorithm (PDF, CDF and QF) with the true random return distributions associated with an optimal policy for a particular state of the environment. Although the PDF and CDF of the random return learnt by the UMDQN-KL and UMDQN-C algorithms are not entirely correct, they remain qualitatively very similar to the true random return distribution, with the multimodality preserved (see blue line). This observation not only validates the soundness of the probability distributions learnt, but also indicates that these two distributional RL algorithms can effectively learn an optimal policy for this benchmark environment. On the contrary, the error made by the UMDQN-W algorithm learning the QF of the random return is much more concerning. In this case, the distributions multimodality is no longer preserved (see blue line). Additional analyses reveal that this difference does not originate from a suboptimal policy learnt by the distributional RL algorithm. In fact, this important observation is consistent with Eq. (16) together with the explanation from Section 4.1 regarding the learning of the QF based on the distributional Bellman operator: the expectation of the random return is preserved, but the probability distribution higher-order moments are not. As previously explained, this analysis is not specific to the UMDQN-W algorithm and applies to several state-of-the-art distributional RL algorithms. To illustrate that claim, Fig. 5 plots the probability distributions of the random return learnt by the CDQN, QR-DQN, IQN and FQF algorithms, which all achieve an optimal policy for the stochastic grid world environment. On the one hand, the CDQN algorithm learning the categorical PDF of the random return based on the KL diver-

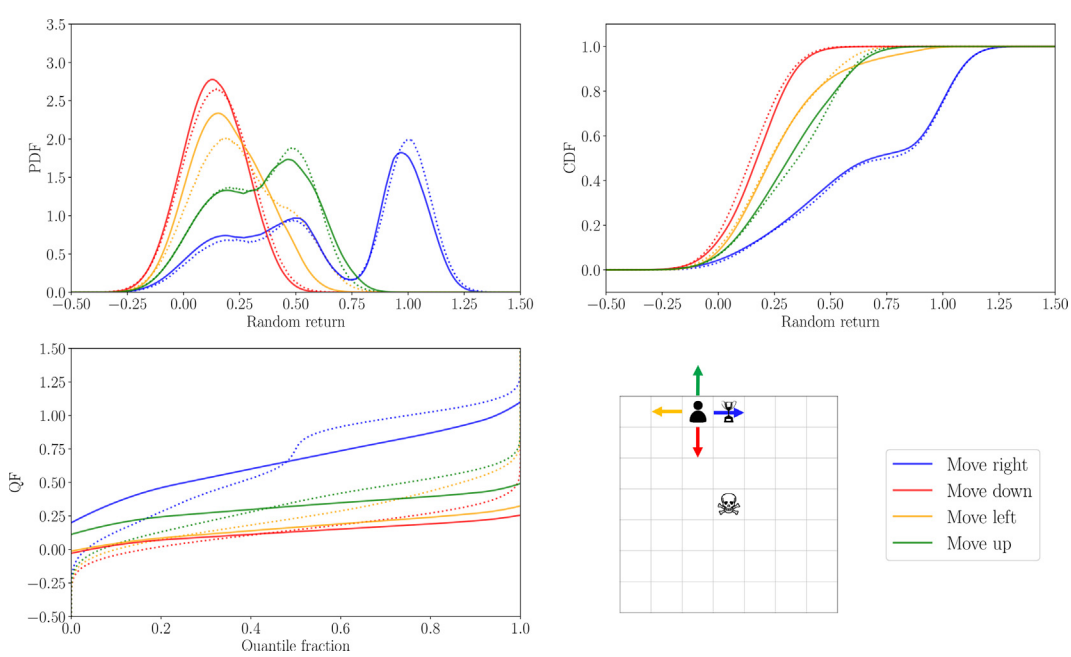


Fig. 4. Comparison of the random return distributions (PDF, CDF and QF) learnt by the UMDQN algorithm (plain lines) with the true random return probability distributions (PDF, CDF and QF) estimated via Monte Carlo and associated with an optimal policy (dotted lines), for a particular state of the stochastic grid world environment.

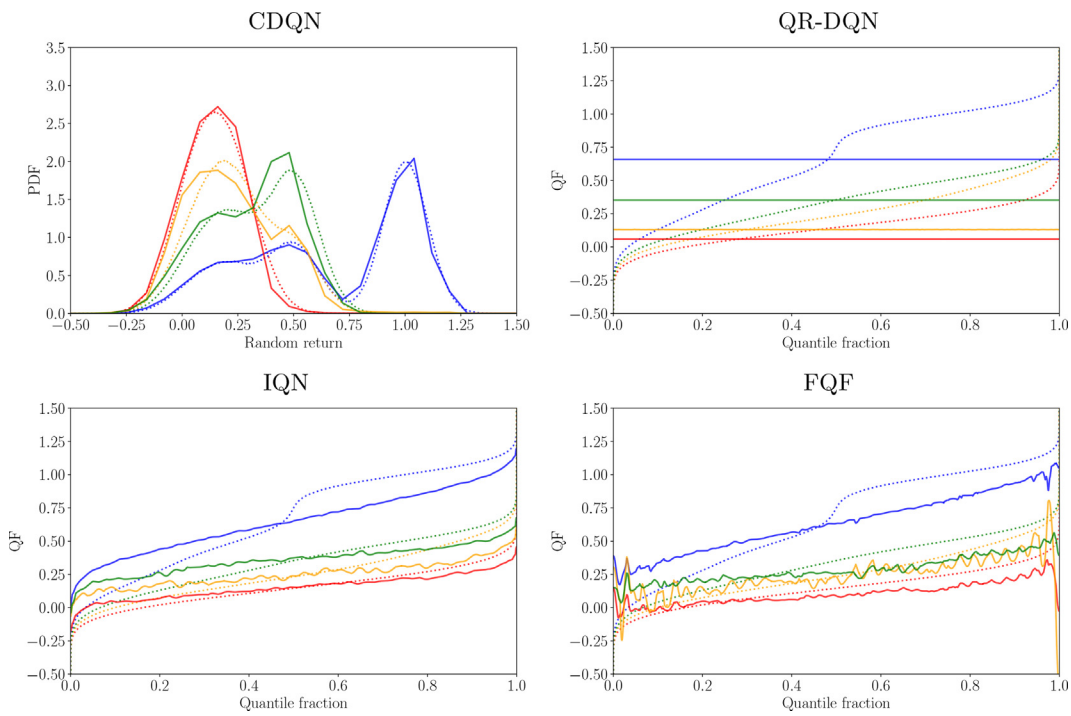


Fig. 5. Comparison of the random return distributions learnt by the CDQN, QR-DQN, IQN and FQF state-of-the-art algorithms (plain lines) with the true random return probability distributions estimated via Monte Carlo and associated with an optimal policy (dotted lines), for a particular state of the stochastic grid world environment.

gence achieves satisfying results, in line with the previous observation for the UMDQN-KL algorithm. On the other hand, the QR-DQN, IQN and FQF algorithms clearly show their limitations for accurately modelling the QF of the random return. Therefore, this particular learning methodology adopted by several state-of-the-art distributional RL algorithms should only be considered when the objective is to learn policies maximising the expectation of the random return, but should instead be discarded when the intention is to exploit the complete probability distribution, for learning risk-aware policies for instance.

Policy performance As far as the quality of the decision-making policies learnt by the distributional RL algorithms is concerned, Fig. 6 presents the results achieved by the three versions of the UMDQN algorithm on the benchmark environments introduced in Section 5.1. The policy performance plotted is the cumulative reward achieved by the RL agent over one episode. For the sake of reliability, the results are averaged over five different random seeds and the variance is highlighted. Moreover, for improved readability, a moving average operation is performed to further smooth the curves. Taking into account their respective strengths and weaknesses detailed below, it is quite difficult to identify a clear winner overall in terms of policy performance among the three versions of the UMDQN algorithm, even though the UMDQN-KL algorithm lags behind the other two. Since the same function approximator class is used, this conclusion also stands for the distribution representations and the probability metrics underneath the distributional RL algorithms. An argument for potentially explaining this observation is the fact that a neural network may more efficiently model the PDF, CDF or QF of the random return distributions depending on the characteristics of these particular probability distributions (multimodality, values of the moments). Another hypothesis is to point out the approximation of the loss defined in Eq. (10), whose effect is not yet clearly understood for the different distribution representations and probability metrics, but also potentially depending on the control problem. This is an important open research question for distributional RL. Therefore, based on these observations, the distribution representation of the random return together with the probability metric should ideally be hyperparameters to be tuned depending on the environment and the control problem at hand. This claim contrasts with the current trend observed in distributional RL research, with the focus being mainly set on the QF and Wasserstein distance, as illustrated by the recent QR-DQN, IQN and FQF algorithms. For this reason, the present research work calls for a reconsideration of all distribution representations and probability metrics for future research in distributional RL.

UMDQN-KL algorithm Even though the distributional Bellman operator $\mathcal{T}^{\pi}$ is not a contraction mapping in the KL divergence, Figs. 4 and 6 empirically show that this probability metric can still lead to the learning of both valuable decision-making policies and relevant random return probability distributions. This observation suggests that the contraction property is not a necessary condition for converging towards the correct random return probability distribution. Nevertheless, the learning process of the UMDQN-KL algorithm has been observed to be fairly less stable compared to other distributional RL algorithms. For several benchmark environments, the learning process may even suddenly stop with the performance entirely collapsing and not recovering, as illustrated in Fig. 6m. Additional experiments suggest that the occurrence of this problematic behaviour for a given environment is strongly tied to the domain $\mathcal{X}$ specified as hyperparameter (lower and upper bounds). A too restrained domain inevitably leads to truncated and hence wrong probability distributions for the random return Z^{π}. On the contrary, if the domain is too wide, it may lead to numerical instabilities in the regions of the domain with almost no mass due to the definition of the KL divergence ($\lim_{x\to 0^+} \log(x) = -\infty$). Appropriately setting the hyperparameters associated with this domain $\mathcal{X}$ may be a particularly challenging task since it is strongly dependent on the control problem and because the probability distribution of the random return Z^{π} may significantly vary with different state-action pairs (s, a) as well as during the learning process. Another interesting observation about the UMDQN-KL algorithm is related to the asymmetry of the KL divergence ($\mathcal{L}_{KL}(A, B) \neq \mathcal{L}_{KL}(B, A)$). Empirically, the learning of valu-

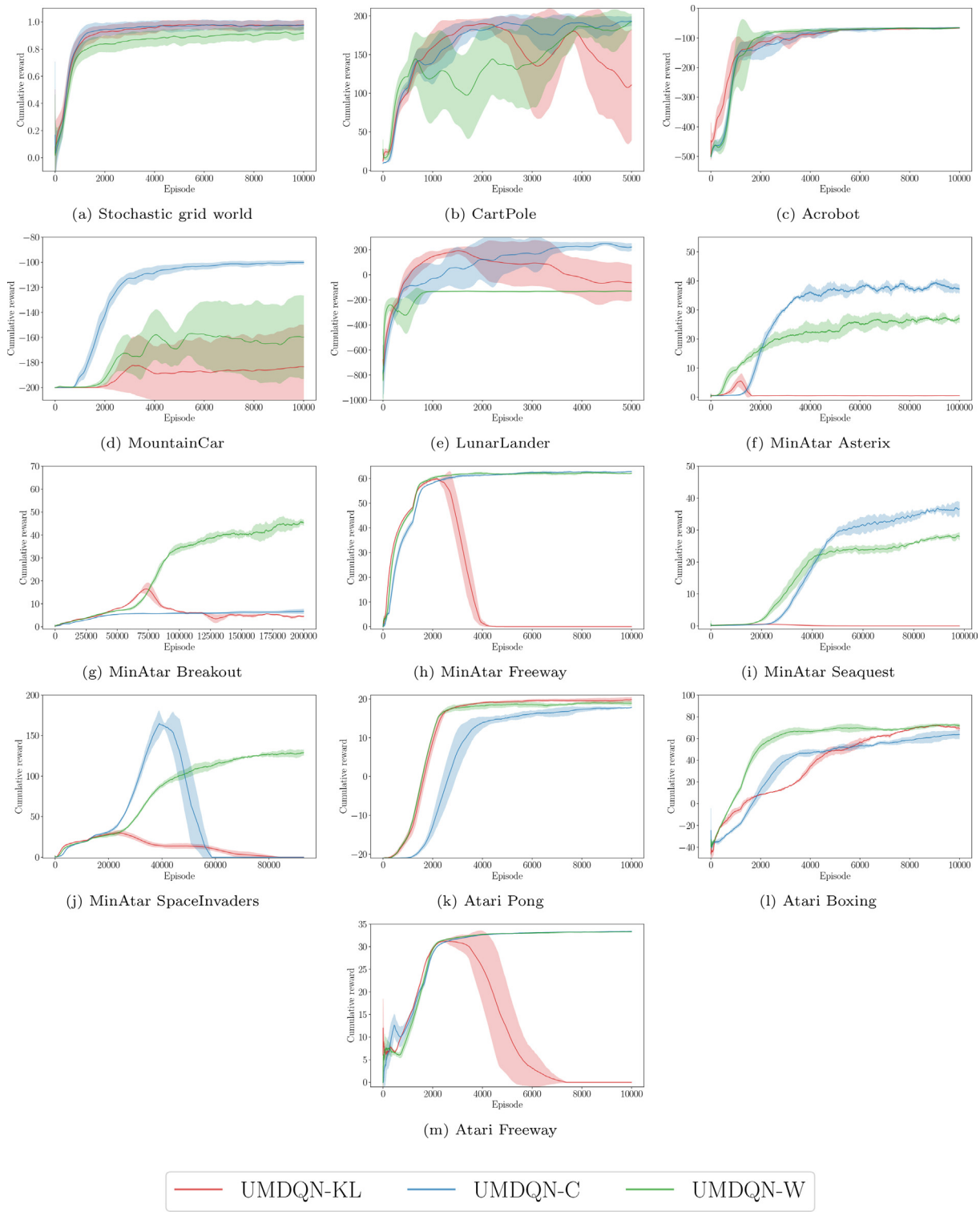


Fig. 6. Performance of the UMDQN algorithm on the benchmark environments proposed.

able policies is observed with the loss $\mathcal{L}_{KL}(\mathcal{T}^{\pi}Z^{\pi}, Z^{\pi})$ but not with $\mathcal{L}_{KL}(Z^{\pi}, \mathcal{T}^{\pi}Z^{\pi})$.

UMDQN-C algorithm Although this distributional RL algorithm also requires the specification of hyperparameters associated with the domain $\mathcal{X}$, it is empirically observed to be far more stable and performing compared to the UMDQN-KL algorithm. This behaviour may potentially be explained by the distributional Bellman operator $\mathcal{T}^{\pi}$ being a contraction in the Cramer distance, but also by the fact that the loss to learn from is symmetric and does not numerically explode around regions of the domain with no probability density. Still, a relevant domain $\mathcal{X}$ has to be specified to expect reliable and satisfying results from the UMDQN-C algorithm, meaning that the range of the returns has to be rigorously approximated beforehand. This requirement is the main weakness of this

particular distributional RL algorithm. In Fig. 6, relevant domains $\mathcal{X}$ are adopted to ensure a fair and interesting comparison. The UMDQN-C algorithm may be the top-performing approach at first glance, but its performance inevitably decreases with less accurate domains.

UMDQN-W algorithm Regarding the performance of the learnt policy, this distributional RL algorithm may probably be the most versatile of the UMDQN algorithms, for two reasons. Firstly, the distributional Bellman operator $\mathcal{T}^\pi$ is a contraction mapping in the Wasserstein distance. Secondly, learning the QF of the random return Z^π does not require the challenging specification of the returns domain $\mathcal{X}$, since the QF takes inputs bounded in the range $[0, 1]$. However, when it comes to the accuracy of the probability distributions learnt, the UMDQN-W algorithm is no longer an acceptable solution, as previously explained in this section. Consequently, this distributional RL algorithm should only be considered for learning decision-making policies maximising the expectation of the random return, but not exploiting the full probability distributions.

For the sake of reproducibility, the complete code used for generating the results presented in this section is made publicly available at the following link: https://github.com/ThibautTheate/Unconstrained-Monotonic-Deep-Q-Network-algorithm. Moreover, the hyperparameters are provided in Appendix E. To end this section, Appendix F briefly compares the policy performance achieved by the UMDQN algorithm with that of the state-of-the-art distributional RL algorithms on the benchmark environments, even though such a comparison is not an objective of this research work. In short, the figure suggests that the results achieved are comparable, which reinforces the soundness of the proposed approach.

6. Conclusions

This research work introduces the *unconstrained monotonic deep Q-network* (UMDQN) distributional RL algorithm, by combining a novel methodology for learning the probability distribution of the random return independently of its representation and the UMNN architecture for modelling these distributions. The experiments performed take advantage of some interesting properties of this novel distributional RL algorithm to yield three important observations. Firstly, the choice of the probability distribution representation coupled with the probability metric has to ideally be dependent on the control problem, since no clear winner could be identified for the set of benchmark environments studied. This result contrasts with the current trend in distributional RL research, which mainly focuses on the QF and Wasserstein distance. Secondly, the methodology adopted by several state-of-the-art algorithms for learning the QF of the random return involves an important approximation, which results in the learning of inaccurate probability distributions. This approach remains totally sound when attempting to learn decision-making policies maximising the expectation of the random return. On the contrary, it should be discarded when aiming to take advantage of other characteristics of the random return distribution, for instance with risk-aware policies. Thirdly, the contraction mapping property for the distributional Bellman operator is not a necessary condition to learn the correct probability distribution of the random return, but may still be beneficial. This highlights the existing gap between theory and practice in distributional RL, and encourages future research on the distributional Bellman operator as well as on the convergence of distributional RL algorithms in general.

To conclude, several avenues are proposed for future work. Firstly, the gap between theory and practice in distributional RL highlighted in this research paper could be narrowed by deriving theoretical guarantees and properties for the novel UMDQN algorithm introduced. Secondly, building on the visualisation and qualitative analysis of the probability distributions presented in this research work, a new performance assessment methodology has to be designed to quantitatively evaluate the accuracy of the random return distributions learnt by a distributional RL algorithm, independently of the resulting policy performance. Indeed, research on distributional RL is generally primarily focused on the latter, neglecting the evaluation of the accuracy of the probability distributions learnt. Thirdly, the approximation in Eq. 16 for the learning of the QF based on the distributional Bellman equation deserves more research, in order to acquire a better understanding of the problem and potentially find an alternative solution. Fourthly, the performance achieved by the UMDQN algorithm is expected to be significantly improved by implementing the diverse enhancements from the Rainbow algorithm [9]: multi-step learning [22], double Q-learning [23], prioritised experience replay [21], duelling architecture [25] and noisy networks [8]. In addition, other more general machine and deep learning techniques could also be investigated, such as [31] or [30] for instance. Lastly, an interesting evolution of the UMDQN algorithm could be to concurrently manage different distribution representations and probability metrics, and intelligently combine this information to further improve the performance of the newly introduced distributional RL algorithm.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Thibaut Théate, Antoine Wehenkel and Adrien Bolland are Research Fellows of the F.R.S.-FNRS, of which they acknowledge their financial support.

Appendix A. Mathematical proofs

This section mathematically supports Eqs. (12), (14) and (16) introduced in Section 4.1. To do so, the relationship between the random variables $Z^\pi(s,a)$ and $\mathcal{T}^\pi Z^\pi(s,a)$ is rigorously determined for different probability distribution representations (PDF, CDF and QF). Proposition 1 and Corollary 1 respectively provide and prove this link for the PDF and CDF of the random return. However, the case of the QF is more complex and involves an approximation, which is discussed at the end of this section.

Proposition 1. *Let $Z^\pi \in \mathcal{Z}$ be the random return associated with the policy $\pi : \mathcal{S} \to \mathcal{A}$, which is a random variable mapping the state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ to the realisation of the return $z \in \mathbb{R}$. Additionally, let $p_R(r|s,a)$ be the probability distribution from which the reward $r \in \mathbb{R}$ is drawn, and $p_T(s'|s,a)$ be the transition probability distribution. Finally, let $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ be the distributional Bellman operator, and let $Z^{\pi'} \in \mathcal{Z}$ be a random variable such that $Z^\pi = \mathcal{T}^\pi Z^{\pi'}$. Then, the probability density functions p_{Z^π} and $p_{Z^{\pi'}}$ associated with the random variables Z^π and $Z^{\pi'}$ respect the following equality:*

$$p_{Z^\pi}(z|s,a) = \mathbb{E}_{\substack{r \sim p_R(\cdot|s,a) \\ s' \sim p_T(\cdot|s,a)}} \left[\frac{1}{\gamma} \, p_{Z^{\pi'}}\left(\frac{z-r}{\gamma} | s', a' \right) |_{a'=\pi(s')} \right]$$
$$\forall z \in \mathbb{R}, \; s \in \mathcal{S}, \; a \in \mathcal{A} \; . \tag{A.1}$$

Proof. Let z be the return sampled from the random variable $Z^\pi(s,a)$ for the state-action pair (s,a). By marginalising over the reward r

collected and over the next state-action pair (s', a') with $a' = \pi(s')$, the PDF of the random return can be expressed as follows:

$$p_{Z^\pi}(z|s,a) = \int p_{Z^\pi}(z|s,a,r,s',a')\ p(r,s',a'|s,a)\ dr\ ds'\ . \tag{A.2}$$

Considering both the conditional independence and the Markov property of the decision-making process, the expression $p(r,s',a'|s,a)$ can be re-written as follows:

$$p(r,s',a'|s,a) = p_R(r|s,a)p_T(s'|s,a)\ . \tag{A.3}$$

According to the distributional Bellman equation, the return z can be expressed as a function of both the reward r and the next return z':

$$z = r + \gamma z'\ . \tag{A.4}$$

Based on this expression and making use of the change of variables theorem, the PDF $p_{Z^\pi}(z|s,a,r,s',a')$ can be re-expressed as follows:

$$p_{Z^\pi}(z|s,a,r,s',a') = |\gamma|^{-1} p_{Z^{\pi'}}(z'|s,a,r,s',a')|_{z'=\frac{z-r}{\gamma}} \tag{A.5}$$

$$= \frac{1}{\gamma}\ p_{Z^{\pi'}}\left(\frac{z-r}{\gamma}|s',a'\right)\ . \tag{A.6}$$

Finally, by substitution of Eqs. (A.3) and (A.6) into (A.2), the following relation is obtained:

$$p_{Z^\pi}(z|s,a) = \int \frac{1}{\gamma}\ p_{Z^{\pi'}}\left(\frac{z-r}{\gamma}|s',a'\right) p_R(r|s,a)p_T(s'|s,a)\ dr\ ds' \tag{A.7}$$

$$= \mathbb{E}_{\substack{r\sim p_R(\cdot|s,a)\\ s'\sim p_T(\cdot|s,a)}}\left[\frac{1}{\gamma}\ p_{Z^{\pi'}}\left(\frac{z-r}{\gamma}|s',a'\right)|_{a'=\pi(s')}\right]\ . \tag{A.8}$$

□

Corollary 1. *Let $Z^\pi \in \mathcal{Z}$ be the random return associated with the policy $\pi : \mathcal{S} \to \mathcal{A}$, which is a random variable mapping the state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ to the realisation of the return $z \in \mathbb{R}$. Additionally, let $p_R(r|s,a)$ be the probability distribution from which the reward $r \in \mathbb{R}$ is drawn, and $p_T(s'|s,a)$ be the transition probability distribution. Finally, let $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ be the distributional Bellman operator, and let $Z^{\pi'} \in \mathcal{Z}$ be a random variable such that $Z^\pi = \mathcal{T}^\pi Z^{\pi'}$. Then, the cumulative distribution functions F_{Z^π} and $F_{Z^{\pi'}}$ associated with the random variables Z^π and $Z^{\pi'}$ respect the following equality:*

$$F_{Z^\pi}(z|s,a) = \mathbb{E}_{\substack{r\sim p_R(\cdot|s,a)\\ s'\sim p_T(\cdot|s,a)}}\left[F_{Z^{\pi'}}\left(\frac{z-r}{\gamma}|s',a'\right)|_{a'=\pi(s')}\right]$$
$$\forall z \in \mathbb{R},\ s \in \mathcal{S},\ a \in \mathcal{A}\ . \tag{A.9}$$

Proof. By considering the definition of the CDF together with Eq. (A.1) given by Proposition 1, the following development can be obtained:

$$F_{Z^\pi}(z|s,a) = \int_{-\infty}^{z} p_{Z^\pi}(z^*|s,a)\ dz^* \tag{A.10}$$

$$= \int_{-\infty}^{z} \mathbb{E}_{\substack{r\sim p_R(\cdot|s,a)\\ s'\sim p_T(\cdot|s,a)}}\left[\frac{1}{\gamma}\ p_{Z^{\pi'}}\left(\frac{z^*-r}{\gamma}|s',a'\right)|_{a'=\pi(s')}\right] dz^* \tag{A.11}$$

$$= \mathbb{E}_{\substack{r\sim p_R(\cdot|s,a)\\ s'\sim p_T(\cdot|s,a)}}\left[\int_{-\infty}^{z} \frac{1}{\gamma}\ p_{Z^{\pi'}}\left(\frac{z^*-r}{\gamma}|s',a'\right)|_{a'=\pi(s')}\ dz^*\right] \tag{A.12}$$

$$= \mathbb{E}_{\substack{r\sim p_R(\cdot|s,a)\\ s'\sim p_T(\cdot|s,a)}}\left[\int_{-\infty}^{\frac{z-r}{\gamma}} p_{Z^{\pi'}}(z^{**}|s',a')|_{a'=\pi(s')}\ dz^{**}\right] \tag{A.13}$$

$$= \mathbb{E}_{\substack{r\sim p_R(\cdot|s,a)\\ s'\sim p_T(\cdot|s,a)}}\left[F_{Z^{\pi'}}\left(\frac{z-r}{\gamma}|s',a'\right)|_{a'=\pi(s')}\right]\ . \tag{A.14}$$

□

As previously mentioned, the case of the QF is more complex and involves an important approximation. Let $Z^\pi \in \mathcal{Z}$ be the random return associated with the policy $\pi : \mathcal{S} \to \mathcal{A}$, which is a random variable mapping the state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ to the realisation of the return $z \in \mathbb{R}$. Additionally, let $p_R(r|s,a)$ be the probability distribution from which the reward $r \in \mathbb{R}$ is drawn, and $p_T(s'|s,a)$ be the transition probability distribution. Finally, let $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ be the distributional Bellman operator, and let $Z^{\pi'} \in \mathcal{Z}$ be a random variable such that $Z^\pi = \mathcal{T}^\pi Z^{\pi'}$. Then, the quantile functions $F^{-1}_{Z^\pi}$ and $F^{-1}_{Z^{\pi'}}$ associated with the random variables Z^π and $Z^{\pi'}$ can be linked based on an approximation as the following:

$$F^{-1}_{Z^\pi}(\tau|s,a) \simeq \mathbb{E}_{\substack{r\sim p_R(\cdot|s,a)\\ s'\sim p_T(\cdot|s,a)}}\left[r + \gamma F^{-1}_{Z^{\pi'}}(\tau|s',a')|_{a'=\pi(s')}\right]$$
$$\forall \tau \in [0,1],\ s \in \mathcal{S},\ a \in \mathcal{A}\ . \tag{A.15}$$

Empirical research on this approximation suggests that it leads to a random variable modelling the quantity Z^π with the correct expectation but potentially different higher-order moments. Moreover, the error resulting from this approximation is observed to increase with the stochasticity characterising the dynamics of the MDP (transition and reward distributions p_T and p_R). On the contrary, Eq. (A.15) no longer relies on an approximation in the deterministic case. As explained in Section 4.1, this particular approximation may have two completely different implications depending on the objective pursued. If the distributional RL algorithm is used to learn ordinary decision-making policies maximising the expectation of the random return, the approach remains totally sound since the probability distribution learnt has the correct first-order moment. On the contrary, this approximation becomes really problematic if the intention is to learn the complete probability distribution of the random return for implementing risk-aware policies.

Appendix B. Implementation details about UMNN

As explained in Section 4.2, the UMNN requires the solving of an integral, which may be a computationally expensive operation. For the sake of efficiency, this integral is numerically computed via the Clenshaw-Curtis quadrature. This technique presents the key advantage of converging exponentially fast for Lipshitz functions. In practice, only a few function evaluations are required for reaching satisfying accuracy, and these operations can be executed in parallel. This approach makes the complete forward computation of the UMNN quite efficient. Regarding the backward pass, the Leibniz rule can be used to make it more memory efficient. This technique enables to compute the derivative of an integral with respect to its inputs as the integral of the derivatives. For the interested reader, more details about the complete implementation of both forward and backward computations can be found in Appendix B of the research paper originally introducing the UMNN architecture [27].

Another operation which has to be efficiently implemented is the expectation of the random return Z^π. Indeed, this important quantity is repeatedly evaluated in the UMDQN algorithm, since the decision-making policy π selects the action maximising the expectation of the random return. The approach implemented for efficiently and accurately estimating the expectation of the random return Z^π is described hereafter for the different versions of the UMDQN distributional RL algorithm.*UMDQN-KL algorithm* As hinted in Section 4.2, the PDF of the random return Z^π is modelled with a UMNN as $p_{Z^\pi}(z) = g(z)\sigma'\left(\int_0^z g(t)dt + \beta\right)$, where the function $\sigma'(\cdot)$ denotes the PDF of a normal distribution (or equivalently

the derivative of a sigmoid function). Consequently, the expectation of the random return Z^π can be expressed as follows:

$$\mathbb{E}[Z^\pi] = \int_{z_{\min}}^{z_{\max}} z g(z) \sigma' \left(\int_0^z g(t)dt + \beta \right) dz \ . \tag{B.1}$$

A straightforward but inefficient solution would be to independently solve each inner integral for different values of the return z. Instead, for improved efficiency, these inner integrals are solved simultaneously by making use of the same neural network evaluation multiple times. The UMNN is first evaluated at evenly separated points between $z_{\min}$ and $z_{\max}$, and a composite Simpson's rule is then applied to approximate the inner integrals. Thereafter, the expectation of the random return Z^π is finally computed by estimating the outer integral using the Monte Carlo approach.

UMDQN-C algorithm Section 4.2 explains that the CDF of the random return Z^π is modelled with a UMNN as $F_{Z^\pi}(z) = \sigma\left(\int_0^z g(t)dt + \beta\right)$, where the function $\sigma(\cdot)$ is a sigmoid function (or equivalently the CDF of a normal distribution). Consequently, the PDF of the random return can be directly derived as $p_{Z^\pi}(z) = g(z)\sigma'\left(\int_0^z g(t)dt + \beta\right)$, and the expectation of the random return Z^π can be evaluated by following the methodology described in the previous paragraph.

UMDQN-W algorithm When the probability distribution of the random return Z^π is represented through the QF, no particular improvement is implemented and the expectation is simply estimated using Monte Carlo, similarly to the state-of-the-art QR-DQN, IQN and FQF distributional RL algorithms.

Appendix C. Implementation details about the UMDQN algorithm

For the sake of completeness and to ease the reader's understanding of the novel approach proposed, this section provides several implementation details and the detailed pseudocodes for the three versions of the UMDQN algorithm presented in this research work. Hence, the UMDQN-KL, UMDQN-C and UMDQN-W are thoroughly explained in Algorithms 2, 3 and 4, respectively.

Modelling the probability distribution of the random return for a terminal state may be tricky and deserves a brief discussion. In this case, the RL agent shall not receive any future rewards, and

Algorithm 2: UMDQN-KL algorithm

Initialise the experience replay memory M of capacity C.
Initialise the main UMNN weights θ (Xavier initialisation).
Initialise the target UMNN weights $\theta^- = \theta$.
for episode = 0 **to** N **do**
  for $t = 0$ **to** T, or until episode termination **do**
    Acquire the state s from the environment $\mathcal{E}$.
    With probability ϵ, select a random action $a \in \mathcal{A}$.
    Otherwise, select $a = \text{argmax}_{a' \in \mathcal{A}} \mathbb{E}[G_Z(s, a'; \theta)]$.
    Interact with the environment $\mathcal{E}$ with action a to get the next state s' and the reward r.
    Store the experience $e = (s, a, r, s')$ in M.
    if $t \% T' = 0$ **then**
      Randomly sample from M a minibatch of N_e experiences $e_i = (s_i, a_i, r_i, s_i')$.
      Derive a discretisation of the domain $\mathcal{X}$ by sampling N_z returns $z \sim \mathcal{U}([z_{\min}, z_{\max}])$.
      for $i = 0$ **to** N_e **do**
        for all $z \in \mathcal{X}$ **do**
          if s_i' is terminal **then**
            Set $y_i(z) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2\right)$ with $\mu = r_i$ and $\sigma = \frac{z_{\max} - z_{\min}}{N_z}$.
          else
            Set $y_i(z) = \frac{1}{\gamma}\ G_Z\left(\frac{z - r_i}{\gamma} | s_i', \text{argmax}_{a_i' \in \mathcal{A}} \mathbb{E}\left[G_Z(s_i', a_i'; \theta^-)\right]; \theta^-\right)$.
          end if
        end for
      end for
      Compute the loss $\mathcal{L}_{KL}(\theta) = \sum_{i=0}^{N_e} \left(\sum_{z \in \mathcal{X}} y_i(z) \log\left(\frac{y_i(z)}{G_Z(z|s_i, a_i; \theta)}\right)\right)$.
      Clip the resulting gradient in the range $[0, 1]$.
      Update the main UMNN parameters θ using the ADAM optimiser.
    end if
    Update the target UMNN parameters $\theta^- = \theta$ every N^- steps.
    Anneal the ϵ-greedy exploration parameter ϵ.
  end for
end for

Algorithm 3: UMDQN-C algorithm

Initialise the experience replay memory M of capacity C.
Initialise the main UMNN weights θ (Xavier initialisation).
Initialise the target UMNN weights $\theta^- = \theta$.
for episode = 0 **to** N **do**
  for $t = 0$ **to** T, or until episode termination **do**
    Acquire the state s from the environment $\mathcal{E}$.
    With probability ϵ, select a random action $a \in \mathcal{A}$.
    Otherwise, select $a = \text{argmax}_{a' \in \mathcal{A}} \mathbb{E}[G_Z(s, a'; \theta)]$.
    Interact with the environment $\mathcal{E}$ with action a to get the next state s' and the reward r.
    Store the experience $e = (s, a, r, s')$ in M.
    if $t \% T' = 0$ **then**
      Randomly sample from M a minibatch of N_e experiences $e_i = (s_i, a_i, r_i, s'_i)$.
      Derive a discretisation of the domain $\mathcal{X}$ by sampling N_z returns $z \sim \mathcal{U}([z_{\min}, z_{\max}])$.
      for $i = 0$ **to** N_e **do**
        for all $z \in \mathcal{X}$ **do**
          if s'_i is terminal **then**
            Set $y_i(z) = \begin{cases} 0 & \text{if } z < r_i, \\ 1 & \text{otherwise.} \end{cases}$
          else
            Set $y_i(z) = G_Z\left(\frac{z - r_i}{\gamma} | s'_i, \text{argmax}_{a'_i \in \mathcal{A}} \mathbb{E}\left[G_Z(s'_i, a'_i; \theta^-)\right]; \theta^-\right)$.
          end if
        end for
      end for
      Compute the loss $\mathcal{L}_C(\theta) = \sum_{i=0}^{N_e} \left(\sum_{z \in \mathcal{X}} (y_i(z) - G_Z(z | s_i, a_i; \theta))^2 \right)^{1/2}$.
      Clip the resulting gradient in the range $[0, 1]$.
      Update the main UMNN parameters θ using the ADAM optimiser.
    end if
    Update the target UMNN parameters $\theta^- = \theta$ every N^- steps.
    Anneal the ϵ-greedy exploration parameter ϵ.
  end for
end for

Algorithm 4: UMDQN-W algorithm

Initialise the experience replay memory M of capacity C.
Initialise the main UMNN weights θ (Xavier initialisation).
Initialise the target UMNN weights $\theta^- = \theta$.
for episode = 0 **to** N **do**
  for $t = 0$ **to** T, or until episode termination **do**
    Acquire the state s from the environment $\mathcal{E}$.
    With probability ϵ, select a random action $a \in \mathcal{A}$.
    Otherwise, select $a = \text{argmax}_{a' \in \mathcal{A}} \mathbb{E}[G_Z(s, a'; \theta)]$.
    Interact with the environment $\mathcal{E}$ with action a to get the next state s' and the reward r.
    Store the experience $e = (s, a, r, s')$ in M.
    if $t \% T' = 0$ **then**
      Randomly sample from M a minibatch of N_e experiences $e_i = (s_i, a_i, r_i, s'_i)$.
      Sample N_τ values for the first quantile fraction $\tau_i \sim \mathcal{U}([0, 1])$.
      Sample N_τ values for the second quantile fraction $\tau_j \sim \mathcal{U}([0, 1])$.
      for $k = 0$ **to** N_e **do**
        for $i = 0$ **to** N_τ **do**
          for $j = 0$ **to** N_τ **do**
            $y_k(\tau_j) = \begin{cases} r_k & \text{if } s'_k \text{ terminal,} \\ r_k + \gamma\, G_Z\left(\tau_j | s'_k,\ \text{argmax}_{a'_k \in \mathcal{A}} \mathbb{E}\left[G_Z(s'_k, a'_k; \theta^-)\right]; \theta^-\right) & \text{otherwise.} \end{cases}$
          end for
          $\delta_{ij}(k) = y_k(\tau_j) - G_Z(\tau_i | s_k, a_k; \theta)$.
        end for
      end for
      Compute the loss $\mathcal{L}_W(\theta) = \sum_{k=0}^{N_e} \left(\sum_{i=0}^{N_\tau} \mathbb{E}_j \left[\rho_{\tau_i}^{\kappa}(\delta_{ij}(k)) \right] \right)$.
      Clip the resulting gradient in the range $[0, 1]$.
      Update the main UMNN parameters θ using the ADAM optimiser.
    end if
    Update the target UMNN parameters $\theta^- = \theta$ every N^- steps.
    Anneal the ϵ-greedy exploration parameter ϵ.
  end for
end for

the random return distribution degenerates into a Dirac distribution shifted by the value of the last reward collected. In practice, such a particular probability distribution may be quite difficult to approximate with a DNN, depending on the distribution representation. Moreover, it may potentially lead to numerical instabilities when computing the loss. For these reasons, this research work makes the choice to smooth out the Dirac distribution whenever appropriate. For the UMDQN-KL algorithm learning a PDF, a normal distribution with a tiny standard deviation is used as a replacement for the problematic Dirac distribution. For the UMDQN-C algorithm which is based on the random return CDF, the step function with infinite slope is supplanted by a smoother version with a large constant slope. Finally, the case of the UMDQN algorithm is left untouched since the QF of a Dirac distribution is trivial to model with a DNN (constant function).

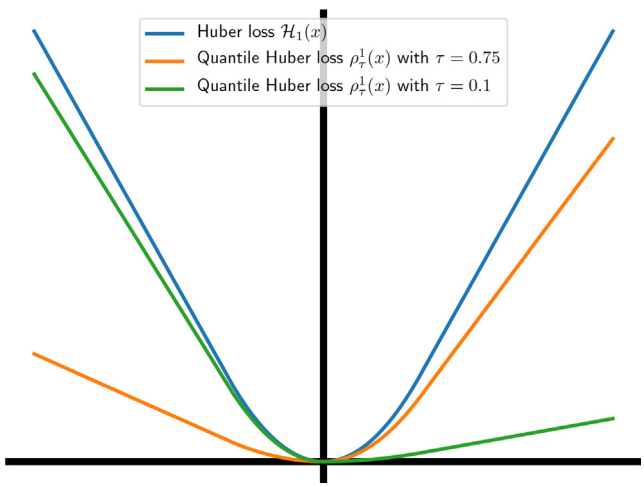


Fig. C.7. Illustration of the quantile Huber loss.

As explained in Section 4.3, the loss defined in Eq. (10) is approximated in the UMDQN algorithm, which may introduce a bias. This problem has already been demonstrated for the Wasserstein distance [1] and a solution has been proposed [7]: the *(conditional) quantile regression* method [10]. Without going into too much detail, this alternative approach is based on the *quantile regression loss*, which is an asymmetric convex loss function respectively penalising overestimation and underestimation errors with weights τ and $1-\tau$, with $\tau \in [0,1]$ being a quantile fraction. This technique is used in the UMDQN-W algorithm, similarly to the state-of-the-art QR-DQN, IQN and FQF distributional RL algorithms. In fact, to ensure smoothness at zero, a slightly modified quantile regression loss is used by these algorithms, the *quantile Huber loss* which is defined for the error $x \in \mathbb{R}$ as follows:

$$\rho_\tau^\kappa(x) = \left|\tau - 1_{\{x<0\}}\right| \frac{\mathcal{H}_\kappa(x)}{\kappa} \ , \tag{C.1}$$

$$\mathcal{H}_\kappa(x) = \begin{cases} \frac{1}{2}x^2 & \text{if} |x| \leqslant \kappa, \\ \kappa(|x| - \frac{1}{2}\kappa) & \text{otherwise}, \end{cases} \tag{C.2}$$

where the threshold κ is a parameter to be tuned. An illustration of the quantile Huber loss with $\kappa = 1$ is provided in Fig. C.7 below. This alternative loss function is evaluated on the pairwise temporal difference (TD) errors δ_{ij} expressed as follows:

$$\delta_{ij} = r + \gamma F_{Z^\pi}^{-1}\left(\tau_j | s', \pi(s')\right) - F_{Z^\pi}^{-1}(\tau_i | s, a) \ . \tag{C.3}$$

Appendix D. Benchmark environments

This section provides an accurate description of the benchmark environments adopted in this research work, together with various implementation details. Additionally, these environments and their associated control problems are illustrated in Figs. D.8, D.9 and D.10.

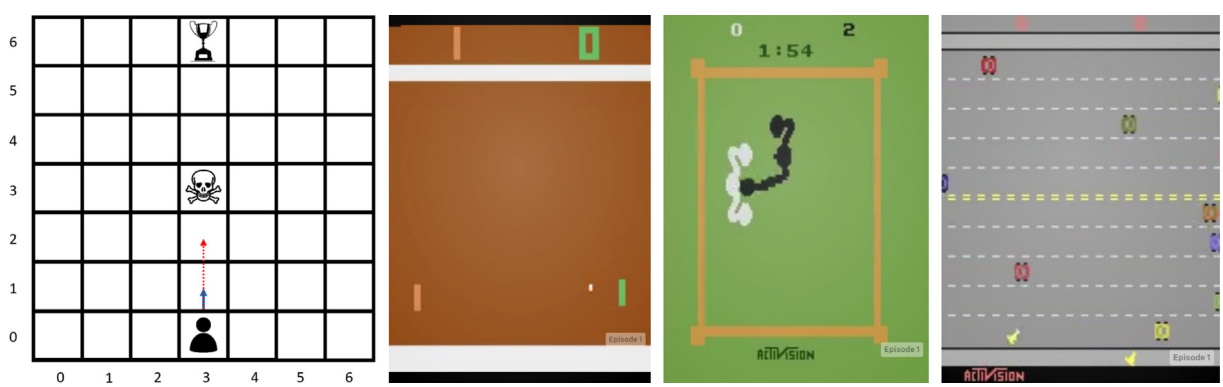


Fig. D.8. Illustration of some benchmark environments with, from left to right, the stochastic grid world and the Atari games Pong, Boxing and Freeway.

Fig. D.9. Illustration of some benchmark environments with, from left to right, the CartPole, Acrobot, MountainCar and LunarLander classic control problems.

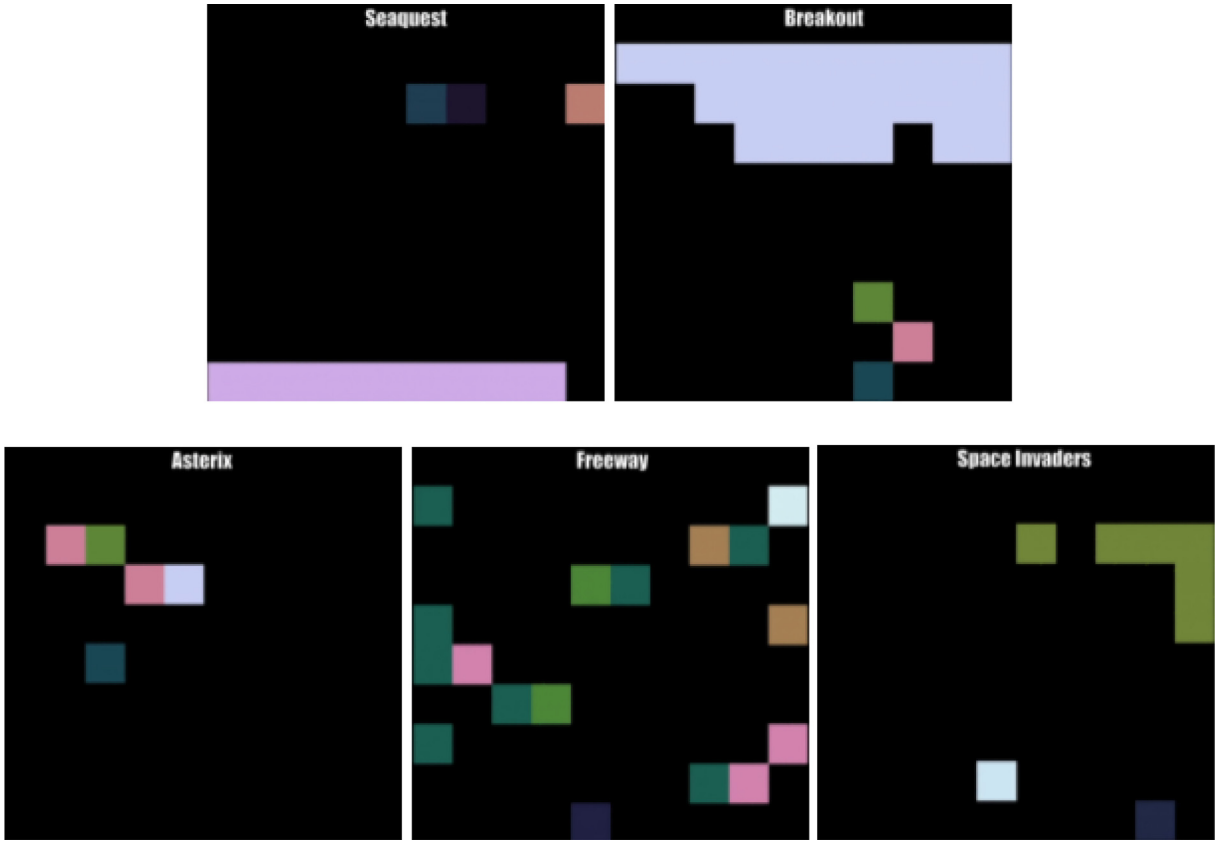


Fig. D.10. Illustration of some benchmark environments with all the MinAtar games.

Stochastic grid world This benchmark environment is a 7×7 grid world, an environment which is commonly considered for analysing and evaluating the performance of RL algorithms. The objective of the agent is simply to reach a certain target location which is fixed, while avoiding a fixed trap. The particularity of this grid world is that both the transition and reward functions are stochastic (p_T and p_R). The intent behind this additional complexity is to better highlight the impact of the distributional RL approach and analyse the probability distributions learnt. The underlying MDP can be defined as follows:

- $\mathcal{S} \in \{0, \dots, 6\} \times \{0, \dots, 6\}$, a state s being composed of the two coordinates of the agent within the grid,
- $\mathcal{A} = \{\texttt{RIGHT}, \texttt{UP}, \texttt{LEFT}, \texttt{DOWN}\}$, with an action a being a moving direction,
- $p_R(r|s,a) \sim \mathcal{N}(\mu, \sigma^2)$ where:
  - $\mu = 1$ if the agent reaches the target location (terminal state),
  - $\mu = -1$ if the agent falls into the trap (terminal state),
  - $\mu = 0$ otherwise,
  - $\sigma = 0.1$ at anytime,
- $p_T(s'|s,a)$ associates a 50% chance to move twice in the chosen direction instead of once, while keeping the agent within the 7×7 grid world (no border crossing allowed),
- p_0 associates an equal probability to all states $s_0 \in \mathcal{S}$, except for the two states corresponding to the trap and target locations which have a null probability,
- $\gamma = 0.5$.

Selection of Atari games This benchmark environment consists of a set of three representative Atari games from the Atari-57 benchmark [2]: Pong, Boxing and Freeway. Similarly to the stochastic grid world, the control problems are made slightly more complex to highlight the impact of the distributional RL approach. Indeed, the deterministic Atari games are made stochastic by using the sticky action generalisation technique (stochastic transitions, but still deterministic rewards). The implementation adopted is the {}`NoFrameskip` from OpenAI gym [4], together with the following wrappers:

- Formatting of a frame to 84×84 pixels,
- Normalisation of the values of the pixels,
- Clipping of the reward to $\{+1,\ 0,\ -1\}$,
- Sending of the episode termination signal when all the agent's lives are lost,
- Execution of a random number of `NOOP` actions at the beginning of an episode (maximum 30),
- Execution of sticky actions with a 0.25 probability,
- Frame skipping and maximisation operation with period 4,
- Stacking of the final 4 frames.

Selection of classic control environments This benchmark environment consists of a set of four classic control problems from the popular OpenAI Gym toolkit [4]: CartPole, Acrobot, MountainCar and LunarLander. Without going into too much detail, these environments can be characterised as follows:

- `CartPole-v0`: The objective is to balance a pole attached by a non-actuated joint to a cart moving along a frictionless track. The state is composed of four continuous values: the cart position, the cart velocity, the pole angle and the pole velocity at the tip. The agent's action is either to push the cart to the left or to

the right. A reward of + 1 is received for each time step with the pole remaining balanced. An episode terminates when the pole angle is more than $\pm 12^\circ$ or when the cart reaches the edge of the display, but also if the episode length is greater than 200.

- `Acrobot-v1`: This system is composed of a double-jointed pendulum, with the joint between the two links being actuated. The objective is to swing the pendulum so that the end of the outer link reaches a given height. The state is a six-dimensional vector describing the system's angles and velocities. To achieve its goal, the agent has three actions at its disposal: either applying no torque, or applying a fixed torque to the left or to the right. The agent is given a reward of -1 for each time step before achieving the objective position. An episode either terminates when this objective is achieved or when the episode length exceeds 500.
- `MountainCar-v0`: The objective is to drive an underpowered car up a steep hill. To achieve that goal, the agent has to learn to leverage potential energy by driving back and forth for gaining momentum. The state consists of both the position and velocity of the car. The agent's action can either be to push the car to the left, do nothing or push the car to the right. A reward of -1 is received at each time step until the goal position is eventually reached. An episode terminates when this particular position is achieved, or if the episode length is greater than 200.
- `LunarLander-v2`: This environment consists of a simulated 2D world within which the objective is to safely land a lander with a limited amount of fuel on a target location. The RL state is composed of 8 values: the two coordinates of the lander, its linear velocities in the horizontal and vertical directions, its angle and angular velocity, as well as two booleans representing whether each leg is in contact with the ground or not. To achieve its objective, the agent has access to four actions: do nothing, fire the left orientation engine, fire the main engine and fire the right orientation engine. A reward between + 100 and + 140 is received for safely landing and coming to rest at the designated location. Additionally, a crash results in receiving a -100 reward, while coming to rest induces a reward of + 100. There is also a + 10 reward generated for each leg with ground contact. Finally, rewards of -0.3 and -0.03 are respectively obtained for each time step firing the main and side engines. The termination of an episode occurs when the lander crashes or gets outside of the viewport, or when the lander is no longer awake (meaning that it does not move nor collide with any other body).

Selection of MinAtar games This benchmark environment consists of a set of five MinAtar games [29]: Seaquest, Breakout, Asterix, Freeway and SpaceInvaders. As explained in Section 5.1, the core objective behind these environments is to make RL experimentation around Atari games more accessible and efficient. To do so, MinAtar reduces the representation complexity of five representative Atari games, while avoiding as much as possible altering the mechanics of the original games. The alternative state representation is of dimension $10 \times 10 \times n$ and binary, where n is the number of channels representing a game-specific object. In addition to other useful features, MinAtar games also include stochasticity in the form of sticky actions and randomised spawn locations, which is particularly important for analysing distributional RL algorithms.

All the benchmark environments used in this research paper are illustrated in Figs. D.8, D.9, D.10. To end this section, it has to be mentioned that this research work makes the choice to evaluate the performance of a decision-making policy by computing the cumulative reward achieved, similarly to previous works in both classical and distributional RL. This approach is sound for the benchmark environments studied thanks to terminal states or a maximum number of steps preventing the performance from indefinitely increasing.

Appendix E. Hyperparameters

This section presents the main hyperparameters adopted for generating the results presented in both Section 5 and Appendix F. The most important criterion taken into account for the selection/tuning of these hyperparameters is the fair comparison between the DRL algorithms studied, while considering at the same time the values reported by the state-of-the-art distribu-

Table E.4
Domain $\mathcal{X}$ set for the different benchmark environments.

Benchmark environment	Lower bound of $\mathcal{X}$	Upper bound of $\mathcal{X}$
Stochastic grid world	-2	2
Atari Pong	-5	5
Atari Boxing	-1	10
Atari Freeway	-1	10
CartPole	-10	110
Acrobot	-110	10
MountainCar	-110	10
LunarLander	-150	200
MinAtar Asterix	-1	10
MinAtar Breakout	-1	10
MinAtar Freeway	-1	10
MinAtar Seaquest	-1	10
MinAtar SpaceInvaders	-1	20

Table E.3
Description of the main hyperparameters associated with the distributional RL algorithms studied.

Hyperparameter	Description
Network structure	Structure of the DNN representing the random return Z^π (neurons per layer).
Discount factor	Discount factor γ adopted for the Q-learning update.
Learning rate	Learning rate of the DL optimiser (ADAM).
Optimiser epsilon	Epsilon of the DL optimiser (ADAM) to improve numerical stability.
Main update frequency	Frequency T_I (in number of steps) at which the main network is updated.
Target update frequency	Frequency N^- (in number of steps) at which the target network is updated.
Replay memory capacity	Capacity C (in number of experiences) of the experience replay memory M.
Batch size	Size of the batch N_e (in experiences) used for each gradient descent iteration.
ϵ-greedy start	Initial value of ϵ, for the ϵ-greedy exploration technique.
ϵ-greedy end	Final value of ϵ, for the ϵ-greedy exploration technique.
ϵ-greedy decay	Exponential decay (in steps) of ϵ, for the ϵ-greedy exploration technique.
ϵ-greedy test	Value of ϵ when testing the policy, for the ϵ-greedy exploration technique.
Number of z values	Number of returns N_z used for representing distributions (PDF and CDF).
Number of τ values	Number of quantile fractions N_τ used for representing distributions (QF).

Table E.5
Hyperparameters selected for the stochastic grid world benchmark environment.

Hyperparameter	UMDQN-KL	UMDQN-C	UMDQN-W
Network structure	$[128]_{\text{DNN}} + [128]_{\text{UMNN}}$	$[128]_{\text{DNN}} + [128]_{\text{UMNN}}$	$[128]_{\text{DNN}} + [128]_{\text{UMNN}}$
Discount factor	0.5	0.5	0.5
Learning rate	10^{-4}	10^{-4}	10^{-4}
Optimiser epsilon	10^{-5}	10^{-5}	10^{-5}
Main update frequency	1	1	1
Target update frequency	1000	1000	1000
Replay memory capacity	10^4	10^4	10^4
Batch size	32	32	32
ϵ-greedy start	1.0	1.0	1.0
ϵ-greedy end	0.01	0.01	0.01
ϵ-greedy decay	10^4	10^4	10^4
ϵ-greedy test	0.001	0.001	0.001
Number of z values	200	200	-
Number of τ values	-	-	200

Table E.6
Hyperparameters selected for the Atari games benchmark environments.

Hyperparameter	UMDQN-KL	UMDQN-C	UMDQN-W
Network structure	$\text{DQN} + [128]_{\text{UMNN}}$	$\text{DQN} + [128]_{\text{UMNN}}$	$\text{DQN} + [128]_{\text{UMNN}}$
Discount factor	0.99	0.99	0.99
Learning rate	5×10^{-5}	5×10^{-5}	5×10^{-5}
Optimiser epsilon	10^{-5}	10^{-5}	10^{-5}
Main update frequency	4	4	4
Target update frequency	10^4	10^4	10^4
Replay memory capacity	10^5	10^5	10^5
Batch size	32	32	32
ϵ-greedy start	1.0	1.0	1.0
ϵ-greedy end	0.01	0.01	0.01
ϵ-greedy decay	10^6	10^6	10^6
ϵ-greedy test	0.001	0.001	0.001
Number of z values	200	200	-
Number of τ values	-	-	200

Table E.7
Hyperparameters selected for the classic control benchmark environments.

Hyperparameter	UMDQN-KL	UMDQN-C	UMDQN-W
Network structure	$[128]_{\text{DNN}} + [128]_{\text{UMNN}}$	$[128]_{\text{DNN}} + [128]_{\text{UMNN}}$	$[128]_{\text{DNN}} + [128]_{\text{UMNN}}$
Discount factor	0.99	0.99	0.99
Learning rate	10^{-4}	10^{-4}	10^{-4}
Optimiser epsilon	10^{-5}	10^{-5}	10^{-5}
Main update frequency	1	1	1
Target update frequency	1000	1000	1000
Replay memory capacity	10^4	10^4	10^4
Batch size	32	32	32
ϵ-greedy start	1.0	1.0	1.0
ϵ-greedy end	0.01	0.01	0.01
ϵ-greedy decay	10^4	10^4	10^4
ϵ-greedy test	0.001	0.001	0.001
Number of z values	200	200	-
Number of τ values	-	-	200

Table E.8
Hyperparameters selected for the MinAtar games benchmark environments.

Hyperparameter	UMDQN-KL	UMDQN-C	UMDQN-W
Network structure	$\text{DQN} + [128]_{\text{UMNN}}$	$\text{DQN} + [128]_{\text{UMNN}}$	$\text{DQN} + [128]_{\text{UMNN}}$
Discount factor	0.99	0.99	0.99
Learning rate	5×10^{-5}	5×10^{-5}	5×10^{-5}
Optimiser epsilon	10^{-5}	10^{-5}	10^{-5}
Main update frequency	4	4	4
Target update frequency	10^4	10^4	10^4
Replay memory capacity	10^5	10^5	10^5
Batch size	32	32	32
ϵ-greedy start	1.0	1.0	1.0
ϵ-greedy end	0.01	0.01	0.01
ϵ-greedy decay	10^6	10^6	10^6
ϵ-greedy test	0.001	0.001	0.001
Number of z values	200	200	-
Number of τ values	-	-	200

tional RL algorithms. First of all, Table E.3 provides a brief description of the hyperparameters to be tuned in the scope of this research work. Then, Table E.4 presents the domain $\mathcal{X}$ selected for the different benchmark environments studied (lower and upper bounds). Finally, Tables E.5, E.6, E.7 and E.8 present the hyperparameters used for generating the results for all the benchmark environments presented in Appendix D.

A complicated choice when it comes to hyperparameters tuning concerns the domain $\mathcal{X}$. Firstly, is it fairer to have the same lower bound $z_{\min}$ and upper bound $z_{\max}$ for the domain $\mathcal{X}$ for all the benchmark environments as it is generally done in scientific literature, or slightly tune these two hyperparameters for each environment? Secondly, if tuned, how to efficiently select relevant values for these two bounds without requiring complicated analyses? The present research work makes the choice to specialise the domain $\mathcal{X}$ for each benchmark environment. This decision is motivated by the diversity of the environments, with completely different ranges for the random return. For each control problem, a quick analysis is performed to estimate the minimum and maximum returns based on the shape of the reward probability distribution p_R.

Let's consider for instance the CartPole environment. Knowing that a + 1 reward is obtained at each time step until episode termination and that the discount factor is equal to 0.99, it is possible to estimate the lower and upper bounds for the returns. In the worst case, the agent totally fails and the episode ends after a few time steps, meaning that the minimum return is close to 0. On the contrary, if the agent manages to continuously keep the pole balanced, the maximum return can be estimated as follows:

$$z_{\max} = \sum_{i=0}^{\infty} 0.99^i \simeq 100 \ .$$

After consideration of a small margin, an appropriate domain $\mathcal{X}$ is obtained for this particular control problem. The same analysis can be repeated for the other benchmark environments, to get Table E.4.

Appendix F. Comparison with state-of-the-art distributional RL algorithms

Even though it is not an objective nor a contribution of the present research work, this section presents a brief comparison of the novel UMDQN algorithm with some state-of-the-art distributional RL algorithms, for the sake of completeness. In particular, the DQN [13], CDQN [1], QR-DQN [7], IQN [6] and FQF [28] are evaluated on the benchmark environments from Section 5.1 and Appendix D alongside the three versions of the UMDQN algorithm. This empirical comparison is presented in Fig. F.11. As explained in Section 5, the results are averaged over five different random seeds for better reliability, and post-processed using the moving average technique to further smooth the curves. However, for the sake of readability, the variances of the distributional RL algorithms are no longer depicted on the plots.

Since the complete Atari-57 benchmark [2] has not been taken into account for evaluating the novel UMDQN distributional RL algorithm proposed, this research work does not make any claim regarding the top-performing approach for this particular benchmark. Still, it is interesting to observe that the UMDQN algorithm achieves an impressive performance which is on par with the state-of-the-art distributional RL algorithms on the benchmark environments adopted in this research paper. Indeed, the UMDQN algorithm consistently ranks in the top three in terms of decision-making policy performance for all these benchmark environments. In the authors' opinion, this result consolidates the soundness of the proposed approach together with the relevance of the conclusions drawn by this research work.

For reproducibility purposes, the complete code executed to generate the results presented in this appendix, including the implementation of the state-of-the-art distributional RL algorithms, is publicly available at the following link: https://github.com/ThibautTheate/Unconstrained-Monotonic-Deep-Q-Network-algorithm. As far as hyperparameters tuning is concerned, values similar to those presented in Appendix E are adopted to ensure a fair comparison between the different distributional RL algorithms.

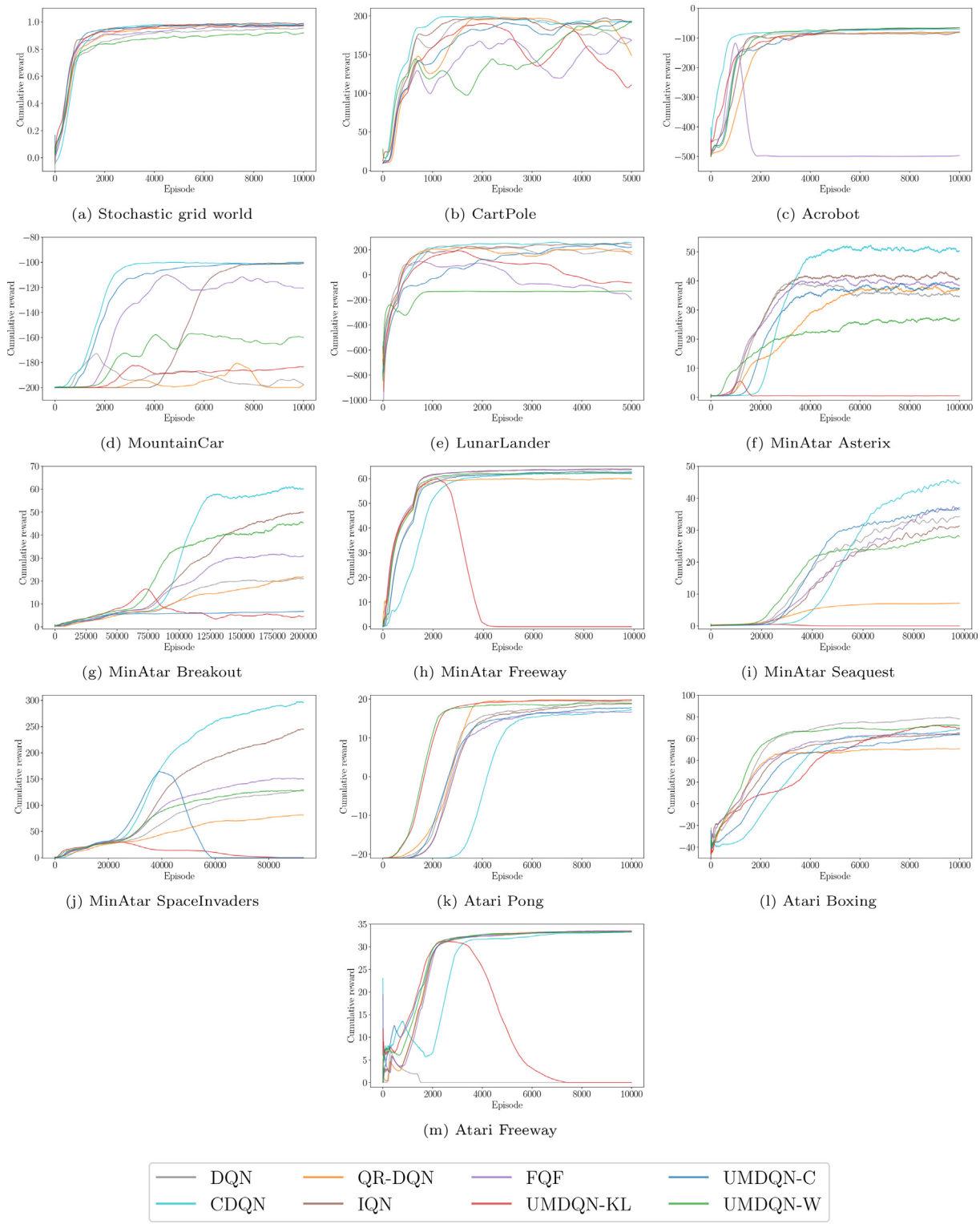


Fig. F.11. Comparison of the UMDQN algorithm performance with the state-of-the-art distributional RL algorithms.

References

[1] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning, ICML, volume 70 of Proceedings of Machine Learning Research, pages 449–458. PMLR, 2017.

[2] Marc G. Bellemare, Yavar Naddaf, Joel Veness, Michael Bowling, The arcade learning environment: an evaluation platform for general agents, J. Artif. Intell. Res. 47 (2013) 253–279.

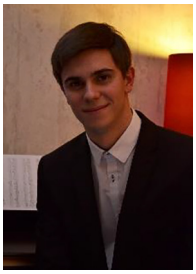
[3] Richard Bellman, Dynamic Programming, Princeton University Press, 1957.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. CoRR, abs/1606.01540, 2016.

[5] Will Dabney, Zeb Kurth-Nelson, Naoshige Uchida, Clara Kwon Starkweather, Demis Hassabis, Rémi Munos, Matthew M. Botvinick, A distributional code for value in dopamine-based reinforcement learning, Nature 577 (2020) 671–675.

[6] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In Proceedings of the 35th

International Conference on Machine Learning, ICML, volume 80 of Proceedings of Machine Learning Research, pages 1104–1113. PMLR, 2018.
[7] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI, pages 2892–2901. AAAI Press, 2018.
[8] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. In 6th International Conference on Learning Representations, ICLR. OpenReview.net, 2018.
[9] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, pages 3215–3222. AAAI Press, 2018.
[10] Roger Koenker, Quantile Regression, Cambridge University Press, 2005.
[11] Clare Lyle, Pablo Samuel Castro, and Marc G. Bellemare. A comparative analysis of expected and distributional reinforcement learning. In AAAI, 2019.
[12] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, Michael Bowling, Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents, J. Artif. Intell. Res. 61 (2018) 523–562.
[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, Demis Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[14] Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. Parametric return density estimation for reinforcement learning. In Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI, pages 368–375. AUAI Press, 2010.
[15] Thanh Nguyen-Tang, Sunil Gupta, and Svetha Venkatesh. Distributional reinforcement learning via moment matching. In AAAI, 2021.
[16] Johan Samir Obando-Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18–24 July 2021, Virtual Event, volume 139 of Proceedings of Machine Learning Research, pages 1373–1383. PMLR, 2021.
[17] Amir Rahimi, Amirreza Shaban, Ching-An Cheng, Richard Hartley, Byron Boots, Intra order-preserving functions for calibration of multi-class neural networks, in: Advances in Neural Information Processing Systems 33, NeurIPS, 2020.
[18] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In Proceedings of the 32nd International Conference on Machine Learning, ICML, volume 37 of JMLR Workshop and Conference Proceedings, pages 1530–1538. JMLR.org, 2015.
[19] Mark Rowland, Marc G. Bellemare, Will Dabney, Rémi Munos, and Yee Whye Teh. An analysis of categorical distributional reinforcement learning. In International Conference on Artificial Intelligence and Statistics, AISTATS, volume 84 of Proceedings of Machine Learning Research, pages 29–37. PMLR, 2018.
[20] Mark Rowland, Robert Dadashi, Saurabh Kumar, Rémi Munos, Marc G. Bellemare, and Will Dabney. Statistics and samples in distributional reinforcement learning. In ICML, 2019.
[21] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In 4th International Conference on Learning Representations, ICLR, 2016.
[22] Richard S. Sutton, Learning to predict by the methods of temporal differences, Mach. Learn. 3 (1988) 9–44.
[23] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pages 2094–2100. AAAI Press, 2016.
[24] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Rui Mao, Makoto Onizuka, Wei Wang, Rui Zhang, Consistent and flexible selectivity estimation for high-dimensional data, CoRR abs/2005.09908 (2020).
[25] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In Proceedings of the 33nd International Conference on Machine Learning, ICML, volume 48 of JMLR Workshop and Conference Proceedings, pages 1995–2003. JMLR.org, 2016.
[26] Christopher J.C.H. Watkins and Peter Dayan. Technical note: Q-learning. Machine Learning, 8:279–292, 1992.
[27] Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. In Advances in Neural Information Processing Systems 32, NeurIPS, pages 1543–1553, 2019.
[28] Derek Yang, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian, and Tie-Yan Liu. Fully parameterized quantile function for distributional reinforcement learning. In Advances in Neural Information Processing Systems 32, NeurIPS, pages 6190–6199, 2019.
[29] Kenny Young, Tian Tian, MinAtar: An Atari-inspired testbed for thorough and reproducible reinforcement learning experiments, CoRR abs/1903.03176 (2019).
[30] Jie Zhang, Yanjiao Li, Wendong Xiao, Zhiqiang Zhang, Non-iterative and fast deep learning: Multilayer extreme learning machines, J. Franklin Inst. 357 (13) (2020) 8925–8955.
[31] Jie Zhang, Wendong Xiao, Yanjiao Li, Sen Zhang, Residual compensation extreme learning machine for regression, Neurocomputing 311 (2018) 126–136.
[32] Fan Zhou, Jianing Wang, and Xingdong Feng. Non-crossing quantile regression for distributional reinforcement learning. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6–12, 2020, virtual, 2020.



Thibaut Théate received his M.Sc. degree in Electrical Engineering in 2018 from the University of Liége, Belgium. He is currently a Ph.D. candidate funded by the F.R.S.-FNRS in the Department of Electrical Engineering and Computer Science of the same university, supervised by Prof. Damien Ernst. His main research interests include Artificial Intelligence and Machine Learning in general, together with their applications to energy markets.

Antoine Wehenkel received his M.Sc. degree in Computer Engineering in 2018 from the University of Liége, Belgium. He is currently a Ph.D. candidate funded by the F.R.S.-FNRS in the Department of Electrical Engineering and Computer Science of the same university, supervised by Prof. Gilles Louppe. His main research interests include Deep generative modelling, information theory and simulation-based inference.

Adrien Bolland received his M.Sc. degree in Electrical Engineering in 2020 from the University of Liége, Belgium. He is currently a Ph.D. candidate funded by the F.R.S.-FNRS in the Department of Electrical Engineering and Computer Science of the same university, supervised by Prof. Damien Ernst. His main research interests include optimal control theory, reinforcement learning and their applications to power systems.

Gilles Louppe is an associate professor in artificial intelligence and deep learning at the University of Liége (Belgium). Previously, he held positions as a research fellow at CERN and as a postdoctoral associate at New York University. His research is at the intersection of deep learning, approximate inference and the physical sciences. He is active in the development of simulation-based inference algorithms based on deep learning, with several applications in the physical sciences.

Damien Ernst received his M.Sc. and Ph.D. degrees in engineering from the University of Liége, Belgium, in 1998 and 2003, respectively. He is currently a Full Professor at the University of Liége, and Visiting Professor at Télécom Paris. His research interests include electrical energy systems and reinforcement learning. He is also the CSO of Haulogy, a company developing intelligent software solutions for the energy sector. He has co-authored more than 300 research papers and two books. He has also won numerous awards for his research and, among which, the prestigious 2018 Blondel Medal.