TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Reliability in Reinforcement Learning and Off-Policy Evaluation

Zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Statistics – Probability – Mathematics in Economics

eingereicht von

Jakob aus der Schmitten
Matrikelnummer: 11901960

ausgeführt am Institut für Information Systems Engineering
der Fakultät für Informatik der Technischen Universität Wien

Betreuer: Associate Prof. Dr.techn. Dipl.-Ing. Clemens Heitzinger

Wien, am 27.03.2025

Jakob aus der Schmitten	Clemens Heitzinger

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 27.03.2025

Jakob aus der Schmitten

Acknowledgments

First, I want to thank Professor Heitzinger for his great supervision on this thesis. I also want to thank Carlotta Tubeuf for the productive and kind collaboration.

Furthermore, this work was supported by the project "RELY – Reliable Reinforcement Learning for Sustainable Energy Systems," funded by the Austrian Research Funding Agency (FFG) under the grant number #FO0999899921. Parts of the calculations were performed using supercomputer resources provided by the Vienna Scientific Cluster (VSC).

Last and definitely not least, I would like to thank my family for supporting me through everything and making my life so much better.

Abstract

This thesis discusses several approaches to distributional reinforcement learning and off-policy evaluation that aim to increase the reliability of reinforcement learning. Furthermore, we discuss convergence guarantees for these algorithms. The corresponding algorithms are applied to two simulation models of a pump turbine, which is part of a pumped storage system and therefore needs to be operated in a reliable manner.

We compare the performance of the algorithms on the environments and discuss challenges regarding the design of the reward function and the implementation of the software. Lastly, we present some ideas on how to compare and analyze the learnt distributions.

Contents

1 Introduction

Alongside supervised and unsupervised learning, reinforcement learning (RL) is often considered as one of the fundamental paradigms of machine learning. In RL, an agent tries to find an optimal strategy for a specific task by interacting with its environment (Sutton and Barto, 2018). Especially in the beginning of training an agent, these interactions are usually quite random to ensure a sufficient amount of exploration. But also after training is finished, it is not necessarily trivial to know how the agent behaves in different situations. While unexpected behaviour might not be a problem when the agent is trained on a simulator, it can have a massively negative impact in other problem settings. Imagine safety-critical applications such as autonomous driving or controlling a certain physical machine. In both cases, an unexpected action can lead to people being injured or major financial losses, for example.

In Chapter 2, we give an introduction into RL and distributional RL by describing the concept of RL and introducing the necessary definitions.

Five distributional RL algorithms are introduced in Chapter 3, which have empirically proven to outperform existing RL approaches, which maximize the expected value of the return (Bellemare et al., 2017; Dabney et al., 2017, 2018; Yang et al., 2019). Furthermore, we present in detail pseudo code for the algorithms, which is not provided in all of the original publications.

In Chapter 4, we discuss convergence guarantees for the algorithms introduced in Chapter 3. Especially, we discuss finite sample convergence guarantees. These can be advantageous in practical applications, as they provide probabilistic error bounds for a finite number of iterations.

It is often not possible in practice that the RL agent directly interacts with the environment. Therefore, we discuss different methods for off-policy evaluation in Chapter 5. These use a collected dataset to evaluate a policy and therefore do not need to interact with the environment.

Another example of a risk sensitive application is the control of a pump turbine used in a pumped hydro storage system. One can't simply let the agent operate on such expensive machinery, as it might cause major financial damage and such critical infrastructure needs to be controlled reliably. More precisely, we present two physical simulation models that respectively act as the environments for the RL algorithms in Chapter 6.

In Chapter 7, the results of the distributional RL algorithms presented in Chapter 3, applied to the pump turbine environments are shown. We discuss the search for well performing hyper parameters and problems we faced during learning. Furthermore, we present methods for using the learned distributions to increase reliability.

2 Expected and Distributional Reinforcement Learning

In this Chapter 2, we give a short introduction into RL. Subsection 2.1 deals with the intuition behind RL, introduces the corresponding notation and basic definitions. The method discussed in this subsection can be considered as the classic form of RL.

In Subsection 2.2, an advancement of classic RL is introduced by formulating the corresponding definitions, fundamental results and comparing it to the method described in Subsection 2.1.

2.1 Notation and Basic Definitions

The environment the RL agent interacts with is represented by a finite, or possibly infinite discrete or continuous, state space $\mathcal{S}$. At each timestep t and corresponding state $s_t \in \mathcal{S}$, based on a policy $\pi(s_t)$, the agent chooses an action a_t in the finite or possibly infinite action space $\mathcal{A}$. This action leads to a new state s_{t+1}. The transition from a state-action pair (s_t, a_t) to another state s_{t+1} is determined by a transition kernel $P(.|s_t, a_t)$. After each action, the agent receives a reward r_t, based on a given reward function R, which output is determined by the state s_{t+1} to which the action a_t led. This interaction of the agent with the environment is depicted in Figure 2.1.

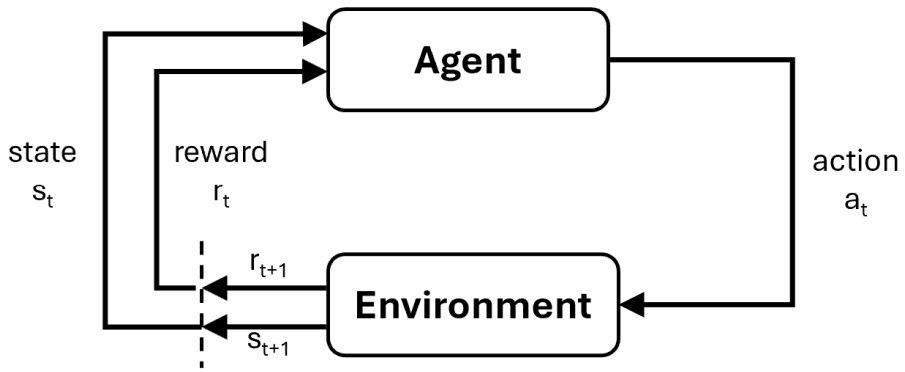


Figure 2.1: Interaction of agent and environment: The agent gets an input from the environment and outputs an action, while the environment takes an action from the agent and outputs a state and a reward (Sutton and Barto, 2018).

The whole procedure is modelled as a Markov Decision Process (MDP), given as

$$(\mathcal{S}, \mathcal{A}, R, P, \gamma).$$

Here, $\gamma \in [0, 1]$ denotes the discount factor. It determines how much a future reward, based on a preceding state and action, is taken into account.

The goal of the agent is to find a policy π that maximizes the action value function

$$Q^\pi(.,.) := \mathbb{E}Z^\pi(.,.). \tag{1}$$

Where the return Z^π is given by the sum of discounted rewards, obtained from starting in state s with action a and thereafter following a policy π (Bellemare et al., 2017), i.e.,

$$Z^\pi(s,a) := \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t). \tag{2}$$

The states and actions at each timestep are given by

$$s_0 = s,\ a_0 = a,\ s_{t+1} \sim P(.|s_t, a_t),\ a_{t+1} \sim \pi(.|s_{t+1}),$$

where $A \sim B$ is understood as A follows the distribution/transition kernel B. Looking at the definition of the return (2), one can see that a discount factor strictly smaller than 1 leads to rewards far in the future having much less impact on the return than rewards in the near future. If the absolute values of the rewards are uniformly bounded by a constant C and $\gamma < 1$, then the return can be bounded by

$$|Z^\pi(.,.)| \leq C \sum_{t=0}^{\infty} \gamma^t = C \frac{1}{1-\gamma}. \tag{3}$$

Furthermore, observe that

$$Q^\pi(s,a) = \mathbb{E}R(s,a) + \gamma \mathbb{E}_{\pi,P}\ Q^\pi(s',a'), \tag{4}$$

where s' is the next state, distributed as $P(.|s,a)$ and the next action a' is distributed as $\pi(.|s)$. Equation (4) is fundamental in RL and widely known as the *Bellman equation* (Bellman, 1957). The action value function can be maximized by solving the optimality equation (Sutton and Barto, 2018)

$$Q^\star(s,a) = \mathbb{E}R(s,a) + \gamma \mathbb{E}_P\ \max_{a' \in \mathcal{A}} Q^\star(s',a'), \tag{5}$$

which has a unique fixed point $Q^\star$. Where $Q^\star$ is the optimal action value function, corresponding to the set of optimal policies $\Pi^\star$. Optimality of a policy is defined as follows.

Definition 1. A policy $\pi^\star$ is *optimal*, i.e., $\pi^\star \in \Pi^\star$, if

$$\mathbb{E}_{a \sim \pi^\star} Q^\star(s,a) = \max_{a \in \mathcal{A}} Q^\star(s,a).$$

Both, the Bellman equation (4) and the optimality equation (5), can be expressed using the Bellman operator $\mathcal{T}^\pi$ and optimality operator $\mathcal{T}$. They are given by

$$\begin{aligned} \mathcal{T}^\pi Q(x,a) &:= \mathbb{E}R(x,a) + \gamma \mathbb{E}_{P,\pi}\ Q\left(x', a'\right), \\ \mathcal{T} Q(x,a) &:= \mathbb{E}R(x,a) + \gamma \mathbb{E}_P\ \max_{a' \in \mathcal{A}} Q\left(x', a'\right) \end{aligned}$$

and are both contraction mappings, which yield the fixed points $Q^\pi(.,.)$ and $Q^\star(.,.)$, respectively.

2.2 Distributional Reinforcement Learning

We start by introducing the definitions and crucial results for distributional reinforcement learning and finish with a comparison to the previously described classical RL approach.

Definitions and Fundamental Results

The method described in Section 2.1 uses the expected value of the return and is called *expected reinforcement learning*. It can be regarded as the classic approach of RL. Another, more recent approach is *distributional reinforcement learning* (DistrRL) and was introduced by Bellemare et al. (2017). Instead of only looking at the expected value of the return, one considers the whole distribution of the return. In contrast to equation (4), one recursively represents the return of a given policy π as

$$Z^\pi(s,a) \overset{d}{=} R(s,a) + \gamma Z^\pi(S',A'), \tag{6}$$

where " $\overset{d}{=}$ " means equality in distribution and $Z^\pi(s,a), R(s,a), Z^\pi(S',A')$ and (S',A') are random variables. This means that the law of the return at the state-action pair (s,a) is the same as the law of the reward R at (s,a) plus the discounted return at the next state-action pair (S',A'). We will use the following definition.

Definition 2. Given a policy π and the corresponding return $Z^\pi(s,a)$ at state-action pair (s,a), we denote the *return distribution* as

$$\eta_\pi(s,a),$$

which means that

$$Z^\pi(s,a) \sim \eta_\pi(s,a).$$

We set $b_{r,\gamma}(x) := r + \gamma x$ and denote the push forward measure for an arbitrary measure μ and a function $f : \mathbb{R} \to \mathbb{R}$ as $f_\#$. So for any Borel set A we have $(b_{r,\gamma})_\# \mu(A) = \mu((A-r)/\gamma)$.

Equation (6) is called *distributional Bellman equation* and one defines the following operator.

Definition 3. For a fixed policy π, the *distributional Bellman operator* $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ is given by

$$\mathcal{T}^\pi \eta(s,a) := \mathbb{E}_{S' \sim P(.|s,a),\ A' \sim \pi(.|S'=s'),\ r \sim \mathcal{P}_R(s,a)} (b_{r,\gamma})_\# \eta(S',A'), \tag{7}$$

where $\mathcal{Z}$ is the space of return distributions with bounded moments and $\mathcal{P}_R(s,a)$ is the distribution of the reward $R(s,a)$.

Using this definition, one can also write the distributional Bellman equation (6) as

$$\eta_\pi = \mathcal{T}^\pi \eta_\pi. \tag{8}$$

Given a policy π, the true distribution η_π is a fixed point of the distributional Bellman equation (8). Since one needs to compare distributions, probability metrics are used. Common choices are the Kullback-Leibler divergence, Wasserstein metric and Cramér distance. A comparison of these three metrics can be found in Théate et al. (2023), for example. We will use the Wasserstein metric at several points in this thesis and therefore define it (Villani, 2008).

Definition 4. Let (X, d) be a Polish space and $p \in [0, \infty)$. The *p-Wasserstein metric* between any two probability measures μ and ν on X is defined as

$$W_p(\mu, \nu) := \left(\inf_{\pi \in \Pi(\mu,\nu)} \int_X d(x, y)^p d\pi(x, y) \right)^{1/p},$$

where $\Pi(\mu, \nu)$ is the set of all couplings of μ and ν.

For real valued probability measures μ and ν the p-Wasserstein metric is of the form

$$W_p(\mu, \nu) = \left(\int_0^1 |F_\mu^{-1}(x) - F_\nu^{-1}(x)|^p dx \right)^{1/p},$$

where F_μ^{-1} and F_ν^{-1} are the quantile functions of μ and ν, respectively. Given two return distributions η and ν, we set

$$\overline{W}_p(\eta, \nu) := \sup_{(s,a) \in \mathcal{S} \times \mathcal{A}} W_p(\eta(s, a), \nu(s, a))$$

and call it the *supremum p-Wasserstein metric*. Bellemare et al. (2017) show that the distributional Bellman operator $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ is a γ-contraction in $\overline{W}_p$, i.e.,

$$\overline{W}_p(\mathcal{T}^\pi \eta, \mathcal{T}^\pi \nu) \leq \gamma \overline{W}_p(\eta, \nu). \tag{9}$$

According to Banach's fixed point theorem, this means that $\mathcal{T}^\pi$ has a unique fixed point. One can see that this fixed point is given by η_π in (6). For $\eta_0 \in \mathcal{Z}$ and $\eta_{t+1} := \mathcal{T}^\pi \eta_t$, this means that the sequence $(\eta_t)_{t \in \mathbb{N}}$ converges to η_π in the supremum p-Wasserstein metric, i.e.,

$$\lim_{t \to \infty} \overline{W}_p(\eta_t, \eta_\pi) = 0.$$

So far in this Section 2.2 we were given a fixed policy π and wanted to calculate the return distribution of this policy for each state-action pair. This setting is called *policy evaluation*. Now, we consider the *policy control* setting, where the goal is to find a policy that maximizes the action value function (1).

Definition 5. A return distribution $\eta^\star$ is *optimal*, if it is the return distribution of an optimal policy.

Notice that an optimal return distribution is generally not unique, as there might be multiple optimal policies with different return distributions.

Before we can define the notion of a distributional Bellman optimality operator, we need to introduce another definition.

Definition 6. Given a return Z, a *greedy policy* for Z is a policy π that maximizes the expectation of Z. We denote the set of greedy policies for Z as

$$\mathcal{G}_Z := \left\{ \pi : \sum_{a \in \mathcal{A}} \pi(a|s) \mathbb{E}Z(s, a) = \max_{a' \in \mathcal{A}} \mathbb{E}Z(s, a') \right\}.$$

Using the notion of a greedy policy we can make the following definition.

Definition 7. Let η be the return distribution of a return Z. An operator $\mathcal{T}$ which implements a greedy selection rule, i.e.,

$$\mathcal{T}\eta = \mathcal{T}^{\pi}\eta \ , \ \pi \in \mathcal{G}_Z \tag{10}$$

is called *distributional Bellman optimality operator*.

Bellemare et al. (2017) show that the distributional Bellman optimality operator $\mathcal{T}$ is not as well behaved as the distributional Bellman operator $\mathcal{T}^{\pi}$ of a policy π. Namely, $\mathcal{T}$ is no contraction in any metric that separates $\mathcal{T}\eta$ and η. Furthermore, not all distributional Bellman optimality operators $\mathcal{T}$ have a fixed point. Even if $\mathcal{T}$ has a fixed point $\eta^{\star} = \mathcal{T}\eta^{\star}$ it is not guaranteed that $(\eta_t)_{t\in\mathbb{N}}$ converges to $\eta^{\star}$, where $\eta_0 \in \mathcal{Z}$ and $\eta_{t+1} := \mathcal{T}\eta_t$.

Advantages of Distributional Reinforcement Learning

Calculating the whole return distribution instead of just an expected value can be beneficial when the return is multimodally distributed. In such a case, the expected value is poor at describing the distribution. Furthermore, Bellemare et al. (2017) claim that DistrRL mitigates the effects of learning from a nonstationary policy. Empirical results of different DistrRL algorithms (Bellemare et al., 2017; Dabney et al., 2017, 2018; Yang et al., 2019) seem to confirm the advantages of the distributional approach, compared to expected RL. The advantageous behaviour and main difference of DistrRL, compared to expected RL, comes into play through function approximation (Lyle et al., 2019). When using tabular methods or linear function approximation, both DistrRL and expected RL are expectation-equivalent, in general.

Furthermore, calculating the entire distribution of the return can give a lot of information about the risks of a given policy. Highly left skewed distributions indicate a high probability of getting a low reward, for example. It is also possible to calculate the variance of the return or the probability of the return being in a certain interval. All of this is important when one looks at safety-critical applications such as self-driving cars or controlling an expensive physical machine, where it is necessary to quantify risks. Also in finance there are risk measures which require to know the whole distribution instead of just the expected value.

3 Distributional Reinforcement Learning Algorithms

We come to the formulation of different DistrRL algorithms. Besides introducing the theory and definitions needed for the respective algorithms, we also provide pseudo-code for each algorithm and discuss the convergence behaviour of certain approaches.

3.1 C51

The C51 algorithm is a DistrRL algorithm and was proposed by Bellemare et al. (2017). We will mainly use the notation from Rowland et al. (2018) as it matches the one in this thesis mostly and the authors deal with both policy evaluation and policy control.

This algorithm belongs to the class of categorical distributional RL (CDRL) algorithms, as it uses a finite number of equally spaced support points $z_1, \dots, z_N$ in the domain of the return distribution together with corresponding weights $p_1, \dots, p_N$ to represent an approximation of the return distribution. The support points, which are often called *atoms* in the literature, are fixed beforehand and the weights are learnt and required to sum to one, i.e., $\sum_{i=1}^{N} p_i = 1$. This means that for each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ the approximate return distribution at time t is represented by

$$\tilde{\eta}_t(s, a) = \sum_{i=1}^{N} \delta_{z_i} p_{t,i}(s, a), \tag{11}$$

where $\delta_\cdot$ denotes the Dirac measure. We denote the set of distributions of the form (11) as $\mathcal{P}$. Note, that this approximation of the distribution is necessary as one can not represent the whole space of probability distributions by a finite set of parameters.

Furthermore, one also approximates the distributional Bellman operator (7), because computing expectations is usually infeasible in practice. Usually, this is the case because of lacking knowledge of the underlying distributions or because of limited computational resources. The approximation is done by sampling a transition $(s_t, a_t, r_t, s_{t+1}, a^\star)$, where $a^\star \sim \pi(.|s_{t+1})$ in policy evaluation and $a^\star = \text{argmax}_{a' \in \mathcal{A}} \mathbb{E}_{R \sim \eta(s_{t+1}, a')} R$ when doing policy control (or categorical Q-Learning as Rowland et al. (2018) call it). The approximate distributional Bellman operator, in the policy evaluation setting, is of the form

$$\begin{aligned} (\hat{\mathcal{T}}^\pi \eta_t)(s_t, a_t) &= (b_{r_t, \gamma})_\# \eta_t(s_{t+1}, a^\star), \\ (\hat{\mathcal{T}}^\pi \eta_t)(s, a) &= \eta_t(s, a) \quad \text{if } (s, a) \neq (s_t, a_t). \end{aligned} \tag{12}$$

It is important to notice that the approximate distributional Bellman operator (12) is in expectation equal to the distributional Bellman operator (7).

After an update, i.e., after the approximate distributional Bellman operator was applied, the obtained distribution is in general not of the form (11) anymore. The reason for this is

that the operator shifts and rescales the previous distribution and in general the supports do not coincide anymore. This issue is solved by using a projection operator $\Pi_{\mathcal{C}}$, defined as

$$\Pi_{\mathcal{C}}\left(\delta_y\right) = \begin{cases} \delta_{z_1}, & \text{if } y \leq z_1, \\ \frac{z_{i+1}-y}{z_{i+1}-z_i}\delta_{z_i} + \frac{y-z_i}{z_{i+1}-z_i}\delta_{z_{i+1}}, & \text{if } z_i < y \leq z_{i+1}, \\ \delta_{z_N}, & \text{if } y > z_N \end{cases} \tag{13}$$

and affinely extending $\Pi_{\mathcal{C}}$ to finite mixtures of Dirac measures. The projection shifts the mass from the space between the atoms proportionally back to the positions of the atoms.

Next, a single step of gradient descent is performed on the loss

$$\mathcal{L}\left(\psi, \tilde{\eta}_t(s_t, a_t), \Pi_{\mathcal{C}}(\hat{\mathcal{T}}^{\pi}\tilde{\eta}_t(s_t, a_t))\right), \tag{14}$$

which is given by the Kullback-Leibler divergence of the prediction $\tilde{\eta}_t(s_t, a_t)$ from the target $\Pi_{\mathcal{C}}(\hat{\mathcal{T}}^{\pi}\tilde{\eta}_t(s_t, a_t))$, with respect to the parameters ψ of $\tilde{\eta}_t(s_t, a_t)$. The parametrization of the new estimate $\tilde{\eta}_{t+1}(s_t, a_t)$ is obtained by utilizing the gradient of the loss. Pseudo-code for the C51 algorithm can be seen in Algorithm 1. Note, that the pseudo-code could be slightly different from the original implementation by Bellemare et al. (2017), but it represents the implementation in the CleanRL Python library (Huang et al., 2022) which we use for our experiments.

CleanRL is an open-source library that contains peer-reviewed Deep RL algorithms. A major advantage of the library are the single file implementations of the algorithms, which make it fairly easy to understand the implementations and adapt them to one's own requirements.

Apart from calculating distributions instead of returns, the algorithmic architecture of the C51 algorithm is similar to the (original) DQN architecture in Mnih et al. (2015). Firstly, this means that each transition gets stored in a replay buffer of a certain size. When the replay buffer is full, the oldest transition in the buffer gets deleted to make space for the new transition. Instead of calculating only one update each timestep, a sample is drawn uniformly at random from the replay buffer. The sample is called *minibatch* and updates are calculated for each element in the sample simultaneously. This method is called *experience replay* and one of its advantages is a greater data efficiency. Furthermore, random sampling breaks correlations between consecutive samples and therefore reduces the variance of the updates. By averaging over the whole batch instead of using a single sample, oscillations or divergence in the parameters are also avoided during learning. Another concept in the DQN architecture is the usage of a separate neural network for generating the target distributions. Every *target network frequency* steps, the parameters of the target network are set to the current parameters of the network for categorical function approximation. This should also lead to a smoother training.

3.2 One-Step Distributional Reinforcement Learning

The distributional Bellman optimality operator (10) is not a contraction in any metric, according to Proposition 1 in Bellemare et al. (2017). This means that convergence is not guaranteed in the control setting, since the distributional Bellman optimality operator does not necessarily have a fixed point. The potential existence of multiple optimal policies is

Algorithm 1 C51 – Categorical DistrRL

Input: $\tilde{\eta}_0(s,a) = \sum_{i=1}^{N} \delta_{z_i} p_{0,i}(s,a) \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}$, Policy π **if** policy evaluation
Output: Approximate [optimal] return distribution $\tilde{\eta}_\pi(s,a)$ $[\tilde{\eta}^\star(s,a)] \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}$
Parameters: Number N of atoms $z_.$, number T of total training steps, start training after *learn start* steps, train every *train frequency* steps, update the target network every *target network frequency* steps
Initialize replay buffer D
Initialize neural network with random weights ψ for categorical function approximation
Initialize target neural network with weights $\phi \leftarrow \psi$
Reset environment to obtain initial state s_0 and set *terminated* = False

for *global step* $= 0, \ldots, T$ **do**
  if *terminated* **then**
    reset environment and set *terminated* = False
  end if
  if (policy evaluation **and** greedy action selection) **then**
    $a_t \sim \pi(.|s_t)$
  else if (policy control **and** greedy action selection) **then**
    $a_t \leftarrow \text{argmax}_{a \in \mathcal{A}} \mathbb{E}_{R \sim \tilde{\eta}_t(s_t,a)}[R]$
  else
    Select action a_t randomly
  end if
  Obtain reward r_t and next state s_{t+1} (or set *terminated* = True)
  Add transition (s_t, a_t, r_t, s_{t+1}) to the replay buffer D
  if *global step* $>$ *learn start* **then**
    if (*global step* **modulo** *train frequency* $= 0$) **then**
      Sample minibatch of transitions from D
      for each element (s_t, a_t, r_t, s_{t+1}) in minibatch **do**
        if policy evaluation **then**
          $a^\star \sim \pi(.|s_{t+1})$
        else if policy control **then**
          $a^\star \leftarrow \text{argmax}_{a \in \mathcal{A}} \mathbb{E}_{R \sim \tilde{\eta}_t(s_{t+1},a)}[R]$
        end if
        $\tilde{\eta}(s_t, a_t) \leftarrow (b_{r_t,\gamma})_{\#} \tilde{\eta}_t(s_{t+1}, a^\star)$
        $\hat{\eta}(s_t, a_t) \leftarrow \Pi_{\mathcal{C}} \tilde{\eta}(s_t, a_t)$
      end for
      Calculate losses $\mathcal{L}\left(\psi, \tilde{\eta}_t(s_t, a_t), \hat{\eta}(s_t, a_t)\right)$ for each transition in minibatch
      Use gradient of summed losses to update ψ
      Use updated ψ to set $\tilde{\eta}_{t+1}(s,a) = \sum_{i=1}^{N} \delta_{z_i} p_{t+1,i}(s,a) \quad \forall (s,a)$
    end if
    if (*global step* **modulo** *target network frequency* $= 0$) **then**
      $\phi \leftarrow \psi$
    end if
  end if
end for

the reason for that. While multiple optimal policies have the same optimal value function in expected RL, they can lead to different distributions in DistrRL. Distributional control algorithms might then mix these distributions up and never converge. Nevertheless, assuming that the optimal policy is unique, DistrRL algorithms can also converge in the control case (Rowland et al., 2018).

A one-step approach, which the authors call one-step distributional RL (OS-DistrRL), for **finite state and action spaces** is introduced by Achab et al. (2023) to overcome the non-convergence issues of CDRL. Denoting the set of probability measures on $\mathbb{R}$ with bounded support as $\mathcal{P}_b(\mathbb{R})$, one can define the following two one-step operators.

Definition 8. For a fixed policy π and state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, the *one-step distributional Bellman operator* $\mathcal{T}^\pi : \mathcal{P}_b(\mathbb{R})^{\mathcal{S}\times\mathcal{A}} \to \mathcal{P}_b(\mathbb{R})^{\mathcal{S}\times\mathcal{A}}$ for any distribution function μ is given by

$$\left(\mathcal{T}^\pi \mu\right)(s,a) = \sum_{s'} P\left(s' \mid s,a\right) \delta_{R(s,a)+\gamma \sum_{a'} \pi(a'|s') \mathbb{E}_{Z\sim\mu(s',a')}[Z]}. \tag{15}$$

Definition 9. For a state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, the *one-step distributional Bellman optimality operator* $\mathcal{T} : \mathcal{P}_b(\mathbb{R})^{\mathcal{S}\times\mathcal{A}} \to \mathcal{P}_b(\mathbb{R})^{\mathcal{S}\times\mathcal{A}}$ for any distribution function μ is given by

$$(\mathcal{T}\mu)(s,a) = \sum_{s'} P\left(s' \mid s,a\right) \delta_{R(s,a)+\gamma \max_{a'} \mathbb{E}_{Z\sim\mu(s',a')}[Z]}. \tag{16}$$

Using these definitions, we formulate the previously described contractivity results for the one-step operators.

Theorem 1. Let π be an arbitrary, fixed policy, then

1. for any $1 \leq p \leq \infty$, the one-step operators $\mathcal{T}^\pi$ and $\mathcal{T}$ are γ contractions in $\overline{W}_p$.
2. the Cramér-projected (13) one-step operators $\Pi_{\mathcal{C}} \circ \mathcal{T}^\pi$ and $\Pi_{\mathcal{C}} \circ \mathcal{T}$ are γ contractions in $\overline{W}_1$.

Utilizing Theorem 1, the authors proof the existence and uniqueness of fixed points. Before these results are stated, we introduce the notion of the value function $V^\pi(.)$ of a policy π, which is given by $V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) Q^\pi(s,a)$. The optimal value function $V^\star(.)$ is given analogously.

Theorem 2. Let π be an arbitrary, fixed policy.

1. Then the unique fixed point ν_π of $\mathcal{T}^\pi$ is given by

$$\nu_\pi(s,a) = \sum_{s'} P\left(s' \mid s,a\right) \delta_{R(s,a)+\gamma V^\pi(s')}, \ \forall (s,a) \in \mathcal{S} \times \mathcal{A}.$$

2. Then the unique fixed point ν_* of $\mathcal{T}$ is given by

$$\nu_*(s,a) = \sum_{s'} P\left(s' \mid s,a\right) \delta_{R(s,a)+\gamma V^*(s')}, \ \forall (s,a) \in \mathcal{S} \times \mathcal{A}.$$

3. If $z_1 \leq R(s,a) + \gamma V^{\pi}(s') \leq z_K$ for all triplets (s,a,s'), then the unique fixed point of $\Pi_{\mathcal{C}} \circ \mathcal{T}^{\pi}$ is $\eta_{\pi} = \Pi_{\mathcal{C}}(\nu_{\pi})$.

4. If $z_1 \leq R(s,a) + \gamma V^{*}(s') \leq z_K$ for all triplets (s,a,s'), then the unique fixed point of $\Pi_{\mathcal{C}} \circ \mathcal{T}$ is $\eta_{*} = \Pi_{\mathcal{C}}(\nu_{*})$.

Notice that the first statement of Theorem 2 implies that OS-DistrRL can not distinguish between two policies with the same value functions, which would yield distinct fixed points in CDRL.

Proofs of Theorem 1 and Theorem 2 can be found in the original work (Achab et al., 2023). For the proof of the third statement in Theorem 2, the following Lemma 1 is used.

Lemma 1. For a discrete distribution q with finite support included in $[z_1, z_N]$, the Cramér projection (13) $\Pi_{\mathcal{C}}$ is *mean preserving*, i.e.,

$$\mathbb{E}_{Z\sim q}[Z] = \mathbb{E}_{Z\sim \Pi_{\mathcal{C}} q}[Z].$$

Proof. Let $z_1, \ldots, z_N$ be a set of atoms, q be a discrete distribution with support included in $[z_1, z_N]$ and $p_1, \ldots, p_N$ the probability mass of the atoms respectively. By the definition of the Cramér projection (13) and the definition of the expected value, we have

$$\begin{aligned}
\mathbb{E}_{Z\sim \Pi_{\mathcal{C}} q}[Z] &= \sum_{k=1}^{N} p_k \sum_{j=1}^{N} \mathbb{1}_{z_j < z_k \leq z_{j+1}} \left(\frac{z_{j+1} - z_k}{z_{j+1} - z_j} z_j + \frac{z_k - z_j}{z_{j+1} - z_j} z_{j+1} \right) \\
&= \sum_{k=1}^{N} p_k \sum_{j=1}^{N} \mathbb{1}_{z_j < z_k \leq z_{j+1}} \underbrace{\left(\frac{z_{j+1} z_j - z_k z_j + z_k z_{j+1} - z_j z_{j+1}}{z_{j+1} - z_j} \right)}_{= z_k} \\
&= \sum_{k=1}^{N} p_k z_k \\
&= \mathbb{E}_{Z\sim q}[Z].
\end{aligned}$$

■

Now, we are ready to use OS-DistrRL to design the corresponding algorithm, which is called "OS-C51" by the authors. The OS-C51 algorithm is quite similar to the C51 algorithm, the main difference lies in the target. Where C51 has

$$\Pi_{\mathcal{C}} \left[(b_{r_t,\gamma})_{\#} \sum_{i=1}^{N} \delta_{z_i} p_{t,i}(s_{t+1}, a^{\star}) \right] = \Pi_{\mathcal{C}} \left[\sum_{i=1}^{N} \delta_{r_t + \gamma z_i} p_{t,i}(s_{t+1}, a^{\star}) \right]$$

as a target, the target of OS-C51 is

$$\Pi_{\mathcal{C}} \left[\sum_{i=1}^{N} \delta_{r_t + \gamma Q_t(s_{t+1}, a^{\star})} \right]. \tag{17}$$

One can see from (17), that the OS-C51 target does not change in case of the existence of multiple optimal policies. Furthermore, the OS-C51 target is computationally more efficient, since it consists of at most two atoms.

In Theorem 4.1 in Achab et al. (2023), the authors show that their "one-step categorical DistrRL" (Algorithm 1 in the paper) converges almost surely in the supremum p-Wasserstein metric $\overline{W}_1(.,.)$, in both the policy control and policy evaluation case. Nevertheless, the one-step categorical DistrRL algorithm is a tabular algorithm, which means that a finite number of states and actions is assumed. Therefore, the authors also introduce the "OS-C51" (Algorithm 2 in the paper), which uses neural networks to approximate the probability mass at the respective atoms and is therefore able to deal with continuous state spaces. They also provide Python code for this algorithm, which is based on the C51 implementation in the cleanRL library (Huang et al., 2022). We give detailed pseudo-code in Algorithm 2.

3.3 QR-DQN

Instead of using a fixed set of atoms and learning the probability mass of each atom to approximate the return distribution, one can also fix a number of equally spread probabilities and learn their location. Exactly this approach is introduced by Dabney et al. (2017).

Fixing a number N of quantiles and denoting a parametric model as $\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^N$, one can denote a quantile distribution $\mu_\theta(.,.)$ as

$$\mu_\theta(s,a) := \frac{1}{N}\sum_{i=1}^{N} \delta_{\theta_i(s,a)} \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{18}$$

Which means that each state-action pair (s,a) gets mapped to a uniform probability distribution supported on $\{\theta_i(s,a) \mid i = 1,\dots,N\}$.

This approach has several advantages compared to CDRL. First, one does not need to know the range of the distribution that should be represented, as the quantiles are by definition between 0 and 1. Especially when the range of the returns varies a lot across the states and actions, this is a huge benefit. Furthermore, one does not need the projection step used in CDRL algorithms, as there are no issues of disjoint supports. Thirdly, the Wasserstein-loss, which is given by the Wasserstein metric between the target and the prediction, can be minimized using quantile regression.

One of the tools needed, is the projection of an arbitrary value distribution μ onto the set of quantile distributions (with a fixed number of quantiles N) $\mathcal{Z}_Q$. This projection is called *quantile projection* and defined as

$$\Pi_{W_1}\mu := \operatorname{argmin}_{Z_\theta \in \mathcal{Z}_Q} W_1(Z_\theta, \mu).$$

For a distribution Y with bounded first moment and a uniform distribution U with support $\{\theta_1,\dots,\theta_N\}$, as it is given in (18), it holds that

$$W_1(Y,U) = \sum_{i=1}^{N} \int_{\frac{i-1}{N}}^{\frac{i}{N}} \left| F_Y^{-1}(\omega) - \theta_i \right| d\omega.$$

Algorithm 2 OS-C51 – One-Step Categorical DistrRL

Input: $\tilde{\eta}_0(s,a) = \sum_{i=1}^{N} \delta_{z_i} p_{0,i}(s,a),\ \forall (s,a) \in \mathcal{S} \times \mathcal{A}$
Output: Approximate optimal return distribution $\tilde{\eta}^{\star}(s,a)$, $\forall (s,a) \in \mathcal{S} \times \mathcal{A}$
Parameters: Number N of atoms $z_\cdot$, number T of total training steps, start training after *learn start* steps, train every *train frequency* steps, update the target network every *target network frequency* steps
Initialize replay buffer D
Initialize neural network with random weights ψ for categorical function approximation
Initialize target neural network with weights $\phi \leftarrow \psi$
Reset environment to obtain initial state s_0 and set *terminated* = False

for *global step* $= 0, \ldots, T$ **do**
  if *terminated* **then**
    reset environment and set *terminated* = False
  end if
  if (greedy action selection) **then**
    $a_t \leftarrow \text{argmax}_{a \in \mathcal{A}} \mathbb{E}_{R \sim \tilde{\eta}_t(s_t,a)}[R]$
  else
    Select action a_t randomly
  end if
  Obtain reward r_t and next state s_{t+1} (or set *terminated* = True)
  Add transition (s_t, a_t, r_t, s_{t+1}) to the replay buffer D
  if *global step* $>$ *learn start* **then**
    if (*global step* **modulo** *train frequency* $= 0$) **then**
      Sample minibatch of transitions from D
      for each element (s_t, a_t, r_t, s_{t+1}) in minibatch **do**
        $Q_t(s_{t+1}, a) \leftarrow \sum_{i=1}^{N} z_i \; p_{t,i}(s,a) \quad \forall a \in \mathcal{A}$
        $\hat{\eta}(s_t, a_t) \leftarrow \Pi_{\mathcal{C}} \left(\delta_{r_t + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a')} \right)$
      end for
      Calculate losses $\mathcal{L}\left(\psi, \tilde{\eta}_t(s_t, a_t), \hat{\eta}(s_t, a_t)\right)$ for each transition in minibatch
      Use gradient of summed losses to update ψ
      Use updated ψ to set $\tilde{\eta}_{t+1}(s,a) = \sum_{i=1}^{N} \delta_{z_i} p_{t+1,i}(s,a) \quad \forall (s,a)$
    end if
    if (*global step* **modulo** *target network frequency* $= 0$) **then**
      $\phi \leftarrow \psi$
    end if
  end if
end for

Denoting $\hat{\tau}_i := \frac{\frac{i-1}{N}+\frac{i}{N}}{2}$, Lemma 2 in Dabney et al. (2017) implies that the values

$$\theta_i = F_Y^{-1}\left(\hat{\tau}_i\right) \text{ for } i = 1, \ldots, N \tag{19}$$

minimze $W_1(Y, U)$. The value of the quantile function $F_Y^{-1}(\tau)$, at a quantile τ, can be characterized as the minimizer of the *quantile regression loss*, given as

$$\mathcal{L}_{QR}^{\tau}(\theta) := \mathbb{E}_{\hat{Y}\sim Y}\left[(\hat{Y}-\theta)(\tau - \delta_{\hat{Y}-\theta<0}\right]. \tag{20}$$

According to (19), this means that $W_1(Z_\theta, \mu)$ is minimized by the values $\{\theta_1, \ldots, \theta_N\}$, that minimize

$$\sum_{i=1}^{N} \mathbb{E}_{\hat{Y}\sim Y}\left[(\hat{Y}-\theta_i)(\tau - \delta_{\hat{Y}-\theta_i<0}\right].$$

The minimizers $\{\theta_1, \ldots, \theta_N\}$ can be calculated by stochastic gradient descent, because the loss gives unbiased sample gradients.

Denoting the infinity-Wasserstein metric as $W_\infty(C, D) := \sup_{\omega\in[0,1]} |F_C^{-1}(\omega) - F_D^{-1}(\omega)|$, it can be shown that the combined operator $\Pi_{W_1}\mathcal{T}^\pi$ is a γ contraction with respect to the supremum infinity-Wasserstein metric. Formally, for a MDP with countable state and action spaces and two arbitrary value distributions Z_1, Z_2 with finite moments, it holds that

$$\overline{W}_\infty\left(\Pi_{W_1}\mathcal{T}^\pi Z_1, \Pi_{W_1}\mathcal{T}^\pi Z_2\right) \leq \gamma \overline{W}_\infty\left(Z_1, Z_2\right). \tag{21}$$

This (21) implies the existence of a unique fixed point of the combined operator $\Pi_{W_1}\mathcal{T}^\pi$. Therefore, repeated application of $\Pi_{W_1}\mathcal{T}^\pi$, or its stochastic approximation, leads to convergence to the fixed point. Since $\overline{W}_p(.,.) \leq \overline{W}_\infty(.,.)$, the convergence result holds for all $p \in [1, \infty]$.

Now, a DistrRL algorithm for policy evaluation can be formulated. It is consistent with the theoretical results obtained before and called *quantile regression temporal difference learning* (QRTD) (Dabney et al., 2017). The basis for this algorithm is temporal difference (TD) learning (Sutton, 1988) and well known in the expected RL setting. In TD learning, the update rule for the evaluation of a policy π is

$$\begin{gathered} V^\pi(s) \leftarrow V^\pi(s) + \alpha\left(r + \gamma V^\pi\left(s'\right) - V^\pi(s)\right), \\ a \sim \pi(\cdot \mid s), r \sim \mathcal{P}_R(s,a), s' \sim P(\cdot \mid s, a) \end{gathered}$$

where $V^\pi(.)$ is the value function corresponding to π. Accordingly, the update rule for QRTD is

$$\begin{gathered} \theta_i(s) \leftarrow \theta_i(s) + \alpha\left(\hat{\tau}_i - \delta_{\{r+\gamma z' < \theta_i(s))\}}\right), \\ a \sim \pi(\cdot \mid s), r \sim \mathcal{P}_R(s,a), s' \sim P(\cdot \mid s, a), z' \sim Z_\theta\left(s'\right). \end{gathered}$$

Where $Z^\pi(.)$ is the return distribution, $\theta_i(s)$ the estimated value of $F_{Z^\pi(s)}^{-1}\left(\hat{\tau}_i\right)$ in state s and Z_θ is a quantile distribution (18).

A policy control version of DistrRL using quantile regression is based on the distributional Bellman optimality operator (10). The architecture is quite similar to the one used for

DQN in Mnih et al. (2015) and the authors call the algorithm quantile regression DQN (QR-DQN). Apart from the fact that the neural networks output N quantiles, the main difference lies in the loss function, which is a *quantile Huber loss* in the case of QR-DQN. Denoting

$$\mathcal{L}_\kappa(u) := \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \leq \kappa, \\ \kappa(|u| - \frac{1}{2}\kappa), & \text{otherwise,} \end{cases}$$

the quantile Huber loss for a quantile $\tau \in [0,1]$ and predetermined hyper parameter $\kappa \geq 0$ is given as

$$\rho_\tau^\kappa(u) := |\tau - \mathbb{1}_{\{u<0\}}| \frac{\mathcal{L}_\kappa(u)}{\kappa}, \tag{22}$$

where $\mathbb{1}_{\{u<0\}}$ denotes the indicator function being one for $u < 0$ and zero otherwise. One can see that the quantile Huber loss is an asymmetric squared loss in the interval $[-\kappa, \kappa]$ and reverts to the quantile regression loss (20) outside of this interval. Pseudo-code for the algorithm we are using is given in Algorithm 3. The implementation used is from the "Stable-Baselines3 – Contrib" Python library (Raffin et al., 2021).

3.4 IQN

While a fixed number of quantiles is learnt in Section 3.3, one could also learn the entire quantile function. This approach is taken in Dabney et al. (2018) and has several advantages, such as being capable of approximating any return distribution if the neural network is large enough and resources for training are sufficient. Furthermore, data efficiency is improved because updates can be done with an arbitrary number of samples, given sufficient compute power. By computing an implicit representation of the return distribution, the class of policies can be expanded to policies π_β on arbitrary *distortion risk measures* β. These policies don't simply maximize the action value functions but also consider some kind of risk. As already mentioned, this is of utmost importance when RL policies control expensive machinery or could have an impact on the well-being of people.

More precisely, the so called *implicit quantile network* (IQN) is trained to reparameterize samples from a base distribution, which could, for example, be the uniform distribution $U(.)$ on the interval $[0,1]$, to the respective quantile values of a target distribution. We denote the quantile function of the distribution μ evaluated at $x \in [0,1]$ as $F_\mu^{-1}(x)$. Formally, for a state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, the state-action quantile function maps $\tau \sim U([0,1])$ (or another base distribution) to a sample $F_{\eta(s,a)}^{-1}(\tau) \sim \eta(s,a)$ from the implicitly defined return distribution $\eta(s,a)$. For a distortion risk measure $\beta : [0,1] \to [0,1]$, the *distorted expectation* of $Z(s,a) \sim \eta(s,a)$ is given by

$$Q_\beta(s,a) := \mathbb{E}_{\tau \sim U([0,1])}[F_{\eta(s,a)}^{-1}(\beta(\tau))], \tag{23}$$

and the corresponding risk-sensitive greedy policy is given as $\pi_\beta(s) = \text{argmax}_{a \in \mathcal{A}} Q_\beta(s,a)$. Now, the sampled temporal difference error for two samples $\tau, \tau' \sim U([0,1])$ and policy π_β at time t is the difference

$$\delta_t^{\tau,\tau'} := r_t + \gamma F_{\eta(s_{t+1},\pi_\beta(s_{t+1}))}^{-1}(\tau') - F_{\eta(s_t,a_t)}^{-1}(\tau).$$

Algorithm 3 QR-DQN

Input: $\tilde{\eta}_0(s,a) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\theta_i(s,a)} \quad \forall (s,a) \in \mathcal{S}\times\mathcal{A}$
Output: Approximate optimal return distribution $\tilde{\eta}^\star(s,a) \quad \forall (s,a) \in \mathcal{S}\times\mathcal{A}$
Parameters: Number N of quantiles $\theta_\cdot(.,.)$, number T of total training steps , start training after *learn start* steps, train every *train frequency* steps, update the target network every *target network frequency* steps, hyper parameter κ for quantile Huber loss
Initialize replay buffer D
Initialize neural network with random weights ψ for calculation of quantiles
Initialize target neural network with weights $\phi \leftarrow \psi$
Reset environment to obtain initial state s_0 and set *terminated* = False

for *global step* $= 0, \ldots, T$ **do**
  if *terminated* **then**
    reset environment and set *terminated* = False
  end if
  if (greedy action selection) **then**
    $a_t \leftarrow \text{argmax}_{a\in\mathcal{A}}\, \mathbb{E}_{R\sim\tilde{\eta}_t(s_t,a)}[R]$
  else
    Select action a_t randomly
  end if
  Obtain reward r_t and next state s_{t+1} (or set *terminated* = True)
  Add transition (s_t, a_t, r_t, s_{t+1}) to the replay buffer D
  if *global step* $>$ *learn start* **then**
    if (*global step* **modulo** *train frequency* $= 0$) **then**
      Sample minibatch of transitions from D
      for each element (s_t, a_t, r_t, s_{t+1}) in minibatch **do**
        $Q(s_{t+1}, a) \leftarrow \frac{1}{N}\sum_{i=1}^{N}\theta_i^\psi(s_{t+1}, a) \quad \forall a \in \mathcal{A}$
        $a^\star \leftarrow \text{argmax}_{a'\in\mathcal{A}}\, Q(s_{t+1}, a')$
        $\mathcal{T}\theta_j \leftarrow r_t + \gamma\ \theta_j^\phi(s_{t+1}, a^\star) \quad \forall i = 1, \ldots, N$
      end for
      Calculate quantile Huber losses $\sum_{i=1}^{N}\sum_{j=1}^{N}\rho_{\hat{\tau}_i}^\kappa(\mathcal{T}\theta_j - \theta_i^\psi(x_t, a_t))$ for each transition in minibatch
      Use gradient of summed losses to update ψ
      Use updated ψ to set $\tilde{\eta}_{t+1}(s,a) = \sum_{i=1}^{N}\delta_{\theta_i^\psi(s,a)} \quad \forall (s,a)$
    end if
    if (*global step* **modulo** *target network frequency* $= 0$) **then**
      $\phi \leftarrow \psi$
    end if
  end if
end for

The IQN loss is given by

$$\mathcal{L}(s_t, a_t, r_t, s_{t+1}) = \frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho_{\tau_i}^{\kappa}(\delta_t^{\tau_i, \tau_j'}), \tag{24}$$

where $\rho_{\tau_i}^{\kappa}(.)$ is the quantile Huber loss (22) and τ_i, τ_j' are independent, identically distributed (i.i.d.) samples from a uniform distribution on $[0, 1]$. A sample based risk-sensitive policy can then be found by approximating $Q_\beta(.,.)$ (23) using K samples $\tilde{\tau}_1, \ldots, \tilde{\tau}_K \sim U([0, 1])$ and calculating

$$\tilde{\pi}_\beta(s) = \text{argmax}_{a' \in \mathcal{A}} \frac{1}{K} \sum_{i=1}^{K} F_{\eta(s,a)}^{-1}(\beta(\tilde{\tau}_i)).$$

Pseudo-code for the IQN algorithm can be seen in Algorithm 4. Again, we use the implementation from the "Stable-Baselines3 - Contrib" Python library (Raffin et al., 2021).

In practice, the input to the quantile function at a certain action is given by the element-wise (Hadamard) product $\odot$ of an embedding of the probability τ and the embedded state s. The state embedding $\iota : \mathcal{S} \to \mathbb{R}^d$ is the same as in the DQN algorithm (Mnih et al., 2015). The probability embedding $\nu : \mathbb{R} \to \mathbb{R}^d$ is given by

$$\nu_j(\tau) = \text{ReLU}\left(\sum_{i=0}^{n-1} \cos(\pi i \tau) w_{ij} + b_j\right), \tag{25}$$

where ReLU is the *rectified linear unit* widely known from neural network architectures and $w_{..}, b_.$ are network parameters. The network parameters of both embedding networks ι and ν are also updated using gradient descent on the IQN loss. To ensure better readability we stick to the notation $F_{\eta(s,a)}^{-1}(\tau)$ instead of $F^{-1}(\iota(s) \odot \nu(\tau), a)$ in the pseudocode 4.

3.5 FQF

Instead of sampling the quantiles randomly and learning the corresponding values of the quantile function, as it is done in Section 3.4, one could also learn the locations of the quantiles. We will refer to these locations as *quantile fractions* in the following, since they partition the interval $[0, 1]$ into N parts. This approach is taken in Yang et al. (2019) and the authors claim that it leads to a better approximation of the return distribution, compared to the approaches with fixed or sampled quantile fractions as described in Section 3.3 and Section 3.4. As there can only be handled a finite number of fractions in practice, the Fully parameterized Quantile Function (FQF) algorithm benefits of being able to utilize this finite number of fractions as good as possible.

Denoting the quantile fractions as $0 = \tau_0 < \tau_1 < \cdots < \tau_N = 1$ and the quantile values as $\theta_0, \ldots, \theta_{N-1}$, the FQF algorithm approximates the distribution of the return at each state-action pair (s, a) as

$$\tilde{\eta}(s, a) := \sum_{i=0}^{N-1} (\tau_{i+1}(s, a) - \tau_i(s, a)) \delta_{\theta_i(s,a)}. \tag{26}$$

Algorithm 4 IQN

Output: Approximate quantile function $F^{-1}_{\eta(s,a)}(.)$ of optimal return distribution $\tilde{\eta}^\star(s,a)$, $\forall (s,a) \in \mathcal{S} \times \mathcal{A}$

Parameters: Numbers N, N', K of samples to be drawn from $U([0,1])$, number T of total training steps, start training after *learn start* steps, train every *train frequency* steps, update the target network every *target network frequency* steps, hyper parameter κ for quantile Huber loss, distortion risk measure β

Initialize replay buffer D

Initialize neural network with random weights ψ for calculation of quantile function

Initialize target neural network with weights $\phi \leftarrow \psi$

Reset environment to obtain initial state s_0 and set *terminated* = False

for *global step* $= 0, \ldots, T$ **do**
  if *terminated* **then**
    reset environment and set *terminated* = False
  end if
  if (greedy action selection) **then**
    Sample τ_i from $U([0,1])$ for $1 \leq i \leq K$
    $a_t \leftarrow \text{argmax}_{a \in \mathcal{A}} \frac{1}{K} \sum_{i=1}^{K} F^{-1}_{\eta(s_t,a)}(\tau_i)$
  else
    Select action a_t randomly
  end if
  Obtain reward r_t and next state s_{t+1} (or set *terminated* = True)
  Add transition (s_t, a_t, r_t, s_{t+1}) to the replay buffer D
  if *global step* $>$ *learn start* **then**
    if (*global step* **modulo** *train frequency* $= 0$) **then**
      Sample minibatch of transitions from D
      for each element (s_t, a_t, r_t, s_{t+1}) in minibatch **do**
        Sample $\tilde{\tau}_1, \ldots, \tilde{\tau}_K$ from $U([0,1])$
        $Q(s_{t+1}, a) \leftarrow \frac{1}{K} \sum_{i=1}^{K} F^{-1}_{\eta(s_{t+1},a)}(\beta(\tilde{\tau}_i)),\ \forall a \in \mathcal{A}$
        $a^\star \leftarrow \text{argmax}_{a' \in \mathcal{A}} Q(s_{t+1}, a')$
        Sample τ_i, τ'_j from $U([0,1])$ for $1 \leq i \leq N$ and $1 \leq j \leq N'$
        $\delta_{i,j} \leftarrow r_t + \gamma F^{-1}_{\eta(s_{t+1},a^\star)}(\tau'_j) - F^{-1}_{\eta(s_t,a_t)}(\tau_i),\ \forall i,j$
      end for
      Calculate losses $\frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho^{\kappa}_{\tau_i}(\delta_{i,j})$ for each transition in minibatch
      Use gradient of summed losses to update ψ
    end if
    if (*global step* **modulo** *target network frequency* $= 0$) **then**
      $\phi \leftarrow \psi$
    end if
  end if
end for

A projection is used to project each quantile function $F^{-1}_{\eta(s,a)}$ onto a staircase function that is supported on $\{\tau_0, \ldots, \tau_N\}$ and $\{\theta_0, \ldots, \theta_{N-1}\}$. The 1-Wasserstein distance

$$W_1(\eta(s,a), \theta, \tau) = \sum_{i=0}^{N-1} \int_{\tau_i}^{\tau_{i+1}} |F^{-1}_{\eta(s,a)}(\omega) - \theta_i| \, d\omega \tag{27}$$

is used to measure the distortion between the approximated quantile function and the true quantile function.

The algorithm works as follows. In each iteration of the algorithm, quantile fractions are computed first. These quantile fractions $\{\tau_0, \ldots, \tau_N\}$, which minimize the 1-Wasserstein distance (27), can be found by utilizing (19). More precisely, according to Proposition 1 in Yang et al. (2019), denoting the 1-Wasserstein loss of $F^{-1}_{\eta(s,a)}$ and the corresponding projected quantile function as

$$W_1(\eta(s,a), \tau) := \sum_{i=0}^{N-1} \int_{\tau_i}^{\tau_{i+1}} \left| F^{-1}_{\eta(s,a)}(\omega) - F^{-1}_{\eta(s,a)}(\hat{\tau}_i) \right| d\omega,$$

it holds that

$$\frac{\partial W_1}{\partial \tau_i} = 2F^{-1}_{\eta(s,a)}(\tau_i) - F^{-1}_{\eta(s,a)}(\hat{\tau}_i) - F^{-1}_{\eta(s,a)}(\hat{\tau}_{i-1}) \quad \forall i \in (0, N) \tag{28}$$

and $\forall \tau_{i-1}, \tau_{i+1} \in [0,1], \tau_{i-1} < \tau_{i+1}, \exists \tau_i \in (\tau_{i-1}, \tau_{i+1})$ s.t. $\frac{\partial W_1}{\partial \tau_i} = 0$.

Let ξ be the weights of the quantile fraction proposal network. Then an iterative application of gradients descent to ξ, according to (28), minimizes the 1-Wasserstein distance.

Having computed the quantile fractions, one can perform a training step on the quantile function network, which parameters we denote as ψ. Denoting the temporal difference error at time t for two probabilities $\hat{\tau}_i, \hat{\tau}_j$ as

$$\delta_t^{ij} = r_t + \gamma F^{-1}_{\eta(s_{t+1}, a_{t+1}), \psi}(\hat{\tau}_i) - F^{-1}_{\eta(s_t, a_t), \psi}(\hat{\tau}_j),$$

one obtains the quantile function network loss similarly to (24).

The pseudo-code for the FQF algorithm is shown in Algorithm 5 and we use a version of [1], which we have modified so that the code more closely resembles the structure of the single file implementations in Huang et al. (2022).

In practice, like in the IQN algorithm, the input to the quantile function at a certain action is given by the element-wise (Hadamard) product $\odot$ of an embedding of the probability τ (25) and the embedded state s.

[1] https://github.com/toshikwa/fqf-iqn-qrdqn.pytorch/tree/11d70bb428e449fe5384654c05e4ab2c3bbdd4cd

Algorithm 5 FQF

Output: Approximate quantile function $F^{-1}_{\eta(s,a)}(.)$ of optimal return distribution $\tilde{\eta}^\star(s,a)$, $\forall (s,a) \in \mathcal{S} \times \mathcal{A}$

Parameters: Number N of quantile fractions, number T of total training steps, start training after *learn start* steps, train every *train frequency* steps, update the target network every *target network frequency* steps, hyper parameter κ for quantile Huber loss

Initialize replay buffer D

Initialize neural network with random weights ψ for approximation of quantile function

Initialize target neural network with weights $\phi \leftarrow \psi$

Initialize neural network Γ with random weights ξ for calculation of quantile fractions

Reset environment to obtain initial state s_0 and set *terminated* = False

for *global step* $= 0, \ldots, T$ **do**
  if *terminated* **then**
    reset environment and set *terminated* = False
  end if
  if (greedy action selection) **then**
    Obtain quantile fractions $\tau_0, \ldots, \tau_N$ for all state-action pairs (s_t, a), where $a \in \mathcal{A}$, from Γ_ξ
    $a_t \leftarrow \text{argmax}_{a \in \mathcal{A}} \sum_{i=0}^{N-1} (\tau_{i+1}(s_t,a) - \tau_i(s_t,a)) F^{-1}_{\eta(s_t,a)} \left(\frac{\tau_i(s_t,a) + \tau_{i+1}(s_t,a)}{2} \right)$
  else
    Select action a_t randomly
  end if
  Obtain reward r_t and next state s_{t+1} (or set *terminated* = True)
  Add transition (s_t, a_t, r_t, s_{t+1}) to the replay buffer D
  if *global step* $>$ *learn start* **then**
    if (*global step* **modulo** *train frequency* $= 0$) **then**
      Sample minibatch of transitions from D
      for each element (s_t, a_t, r_t, s_{t+1}) in minibatch **do**
        Obtain quantile fractions $\tilde{\tau}_0, \ldots, \tilde{\tau}_N$ for $(s_{t+1}, a), a \in \mathcal{A}$ from Γ_ξ
        $\hat{\tilde{\tau}}_i(s_{t+1}, a) \leftarrow \frac{\tilde{\tau}_i(s_{t+1},a) + \tilde{\tau}_{i+1}(s_{t+1},a)}{2}$
        $Q(s_{t+1}, a) \leftarrow \sum_{i=0}^{N-1} (\tilde{\tau}_{i+1}(s_{t+1},a) - \tilde{\tau}_i(s_{t+1},a)) F^{-1}_{\eta(s_{t+1},a)} \left(\hat{\tilde{\tau}}_i(s_{t+1},a) \right) \quad \forall a \in \mathcal{A}$
        $a^\star \leftarrow \text{argmax}_{a' \in \mathcal{A}} Q(s_{t+1}, a')$
        $\delta_{i,j} \leftarrow r_t + \gamma F^{-1}_{\eta(s_{t+1},a^\star)}(\hat{\tilde{\tau}}_i(s_{t+1},a^\star)) - F^{-1}_{\eta(s_t,a_t)}(\hat{\tilde{\tau}}_j(s_{t+1},a^\star)) \quad \forall i,j$
      end for
      Calculate losses $\frac{1}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \rho^\kappa_{\left(\hat{\tilde{\tau}}_j(s_{t+1},a^\star)\right)}(\delta_{i,j})$ for all minibatch elements
      Use gradient of summed losses to update ψ and $\frac{\partial W_1}{\partial \tau_i}$ to update ξ
    end if
    if (*global step* **modulo** *target network frequency* $= 0$) **then**
      $\phi \leftarrow \psi$
    end if
  end if
end for

4 Convergence Guarantees

Although we already mentioned some convergence results for DistrRL algorithms in Chapter 3, we will now discuss convergence results of the corresponding algorithms, or modifications of them, in more detail.

Convergence of a Categorical Distributional Reinforcement Learning Algorithm

In Rowland et al. (2018), a modified version of the C51 algorithm (1) is presented. Instead of updating the return distribution by performing gradient descent utilizing the loss (14) given by the Kullback-Leibler divergence, the return distribution is updated using a mixture of the target and the prediction, i.e.,

$$\tilde{\eta}_{t+1}(s,a) = (1-\alpha_t(s,a))\ \tilde{\eta}_t(s,a) + \alpha_t(s,a)\ \hat{\eta}_t(s,a), \tag{29}$$

for all state-action pairs (s,a). Where $\alpha_t(s,a) \in [0,1]$ is a learning rate that depends on the state-action pair (s,a) and time t. Furthermore, $\alpha_t(s,a) = 0$ for $(s,a) \neq (s_t,a_t)$.

Rowland et al. (2018) show in Proposition 1 that the heuristic projection $\Pi_{\mathcal{C}}$ is the orthogonal projection (13) onto the set of categorical distributions $\mathcal{P}$ (11) with respect to the Cramér metric $\ell_2(.,.)$.

Definition 10. Denoting the cumulative distribution functions of two distributions $\eta(.,.)$ and $\mu(.,.)$ as $F_{\eta(.,.)}$ and $F_{\mu(.,.)}$, the *Cramér metric* of two return distributions $\eta(.,.)$ and $\mu(.,.)$ is given by

$$\ell_2(\eta(.,.),\mu(.,.)) = \left(\int_{\mathbb{R}} (F_{\eta(.,.)}(x) - F_{\mu(.,.)}(x))^2 dx\right)^{1/2}.$$

The *supremum-Cramér metric* is defined as

$$\bar{\ell}_2(\eta,\mu) := \sup_{(s,a)\in\mathcal{S}\times\mathcal{A}} \ell_2(\eta(s,a),\mu(s,a)).$$

This implies that $\Pi_{\mathcal{C}}$ is a non-expansion with respect to $\ell_2(.,.)$ also with respect to the supremum-Cramér metric $\bar{\ell}_2(.,)$, if $\mathcal{S}\times\mathcal{A}$ is finite. It is shown in Proposition 2, that the distributional Bellman operator $\mathcal{T}^\pi$ is a $\sqrt{\gamma}$-contraction in $\bar{\ell}_2(.,.)$. Combining these two results one obtains that $\Pi_{\mathcal{C}}\mathcal{T}^\pi$ is a $\sqrt{\gamma}$-contraction in $\bar{\ell}_2(.,.)$. So repeated application of the composition of the projection and the distributional Bellman operator leads to convergence to a unique distribution $\eta_{\mathcal{C}}$ in the supremum-Cramér metric $\bar{\ell}_2(,)$. More formally, this means that for an arbitrary probability distribution $\eta_0(.,.)$, it holds that

$$(\Pi_{\mathcal{C}}\mathcal{T}^\pi\eta_0)^m \overset{m\to\infty}{\longrightarrow} \eta_{\mathcal{C}}$$

in $\bar{\ell}_2$.

In Theorem 1, the authors show that the modified version of the **tabular** C51 algorithm (29) converges to $\eta_{\mathcal{C}}$ almost surely for policy evaluation. Denoting the true distribution function as η_π, it holds that the chosen approximation leads to the error

$$\bar{\ell}_2^2(\eta_\pi, \eta_{\mathcal{C}}) \leq \frac{1}{1-\gamma} \max_{1 \leq i \leq N} (z_{i+1} - z_i), \tag{30}$$

as long as $\eta_\pi(s,a)$ is supported on $[z_1, z_N]$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. This means that the true return distribution can be approximated arbitrarily well by increasing the number of support points $z_.$.

Assuming that the optimal policy is **unique**, it is shown in Theorem 2 that policy control converges to some limit $\eta_{\mathcal{C}}^\star$ almost surely. The greedy policy with respect to the limit-distribution $\eta_{\mathcal{C}}^\star$ is the optimal policy. The distribution $\eta_{\mathcal{C}}^\star$ is in the space of categorical distributions, i.e., $\eta_{\mathcal{C}}^\star \in \mathcal{P}$.

For both Theorem 1 and Theorem 2 to hold, the learning rates $\alpha_.(.,.)$ need to satisfy the the *Robbins-Monro conditions*. This means that

$$\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty,$$
$$\sum_{t=0}^{\infty} \alpha_t(s,a)^2 < C < \infty$$

hold almost surely for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.

Finite-Sample Analysis

Even though it is good to have convergence guarantees for the algorithm one uses, it is even more interesting for practical applications how fast these algorithms converge, i.e., how many data points and iterations are required to obtain an error below some threshold. This topic is often called *complexity* or *finite-sample analysis*.

For a γ-discounted, infinite-horizon, **tabular** MDP in the **synchronous** setting such a non-asymptotic convergence bound can be found for **policy evaluation** (Peng et al., 2024). Where synchronous means that updating the parameters of the model is done in a synchronous manner, if multiple agents/environments are used for training. It is assumed that in each iteration t of the algorithm, a triple $(a_t(s), s_t(s), r_t(s))$ can be generated, according to

$$a_t(s) \sim \pi(.|s), s_t(s) \sim P(.|s, a_t(s)), r_t(s) \sim \mathcal{P}_R(s, a_t(s)),$$

for all states $s \in \mathcal{S}$.

Similar to (29), a modified version of the C51 algorithm is used. Given an initial distribution $\eta_0 \in \mathcal{P}$, the update scheme is of the form

$$\eta_{t+1} = (1 - \alpha_{t+1})\eta_t + \alpha_{t+1} \Pi_C \mathcal{T}_{t+1} \eta_t.$$

Where $\mathcal{T}_t$ is the approximate distributional Bellman operator (12). Since synchronous updates are used, we can write $\eta_.$ instead of $[\eta_.(s)]_{s \in \mathcal{S}}$. Once more, let $\eta_{\mathcal{C}} \in \mathcal{P}$ be the unique fixed point of the composition of the projection onto $\mathcal{P}$ and the distributional Bellman operator.

Peng et al. (2024) present the finite-sample convergence result, which is of the following form.

Theorem 3 (Peng et al. (2024))**.** Denote the number of atoms on which the approximate return distribution is supported as $N+1 \in \mathbb{N}$. Let $\delta \in (0,1), \varepsilon \in (0,1)$ and suppose that $N > \frac{4}{\varepsilon^2(1-\gamma)^3}$. Furthermore, let T be the total number of update steps satisfying

$$T \geq \frac{C_1 \log^3 T}{\varepsilon^2(1-\gamma)^3} \log \frac{|\mathcal{S}|T}{\delta}$$

for some large universal constant $C_1 > 1$, i.e., $T = \widetilde{O}\left(\frac{1}{\varepsilon^2(1-\gamma)^3}\right)$. Where $|\mathcal{S}|$ is the number of elements, i.e., the number of states, in $\mathcal{S}$. Assume that the step size α_t at time t satisfies

$$\frac{1}{1+\frac{c_2(1-\sqrt{\gamma})t}{\log t}} \leq \alpha_t \leq \frac{1}{1+\frac{c_3(1-\sqrt{\gamma})t}{\log t}}$$

for some small universal constants $c_2 > c_3 > 0$. Then, it holds that

$$\bar{W}_1\left(\eta_T, \eta_{\mathcal{C}}\right) \leq \varepsilon$$

with probability at least $1-\delta$.

Convergence of QRDQN

Looking at Quantile Temporal-Difference Learning, it can be shown that the approximating sequence of i-th quantiles $(\theta_t(s,a,i))_{t=0}^{\infty}$ at state-action pair (s,a) converges to the set of fixed points of the projected distributional Bellman operators, with probability 1. This is shown by Rowland et al. (2024) in Theorem 5.1, using the ODE method for stochastic approximation. Again, the result is only shown for the case of **finite state-action spaces**. Furthermore, it needs to be assumed that the return distributions have finite mean.

5 Off-Policy Evaluation

The classic approach of RL is to evaluate or learn a policy by interacting with the environment, as described in Chapter 2 and Chapter 3. As the agent might take random actions especially at the beginning of the training procedure, direct interaction with the environment is a problem or entirely impossible in certain settings. An example is the treatment of patients in intensive care units suffering from sepsis. It is not possible to try out different, random treatment methods in order to perhaps learn a good policy for this task. Especially, it is not possible to try out a new policy, in order to evaluate it. Another example is the control of an expensive machinery. In the absence of a reliable simulation model of the machinery, it is not possible to perform random actions, as these could result in significant damage to the machinery.

In these situations, where direct interaction with the environment is not possible, off-policy evaluation (OPE) can be a viable method to evaluate a policy. We introduce definitions which are fundamental for OPE and the concept of importance sampling in Section 5.1. Furthermore, an approach of lower bounding the expected value of the return of a given policy is discussed in Section 5.2, while a distributional variant of OPE is presented in Section 5.3.

5.1 Fundamental Methodology and Terminology

Let π denote the policy that should be evaluated. In the context of OPE, it is often called the *evaluation policy*. Furthermore, let π_b denote the so-called *behavior policy* and let H denote the horizon of the MDP, which we assume to be finite at first. OPE algorithms only use *trajectories*

$$\tau := (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_H)$$

generated by the behavior policy π_b to estimate the expected value of the return $Z^\pi(.,.)$ of the evaluation policy π. Denoting the distribution of the initial state s_0 as $P_0(.)$, this means that the trajectory is distributed as

$$\tau \sim P_0(s_0) \prod_{t=0}^{H-1} \pi_b(a_t|s_t) P(s_{t+1}|s_t, a_t).$$

The goal is to estimate $\mathbb{E}Z^\pi$, but only trajectories according to the behavior policy π_b and therefore only returns Z^{π_b} are given. In general, it holds that $\mathbb{E}Z^\pi \neq \mathbb{E}Z^{\pi_b}$ and therefore one needs to account for this discrepancy using *importance sampling ratios*. Before we define the latter, an assumptions needs to be made.

Assumption 1. We assume that

$$\pi(a|s) > 0 \implies \pi_b(a|s) > 0,$$

which is often called the *assumption of coverage* (Sutton and Barto, 2018).

Definition 11. Given an evaluation policy π, a behavior policy π_b and a trajectory τ, the *importance sampling ratio*

$$\rho_\tau := \frac{P_0(s_0)\prod_{t=0}^{H-1}\pi(a_t|s_t)P(s_{t+1}|s_t,a_t)}{P_0(s_0)\prod_{t=0}^{H-1}\pi_b(a_t|s_t)P(s_{t+1}|s_t,a_t)}$$

is the relative probability of the trajectory under the evaluation and target policy.

Therefore, the approach is called *importance sampling*. Notice that

$$\rho_\tau = \prod_{t=0}^{T-1}\frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)}, \tag{31}$$

which implies that the importance sampling ratio does not depend on the transition probabilities of the MDP, but only on the trajectory and the evaluation and behaviour policies. Utilizing the importance sampling ratio, we obtain

$$\mathbb{E}Z^\pi = \mathbb{E}\left[\rho_\tau Z^{\pi_b}\right]$$

and are therefore able to estimate the expected return of the policy π by only using data (trajectories) generated by the behaviour policy.

Nevertheless, looking at (31), one can see that large differences in the policies, can lead to very large or diminishing importance sampling ratios. In practice one could approximate the expected value by the arithmetic mean, i.e., one would use multiple trajectories $\tau_1, \dots, \tau_n$ and compute

$$\hat{Q}^\pi := \frac{1}{n}\sum_{i=1}^{n}\rho_{\tau_i}Z^{\pi_b} \approx \mathbb{E}Z^\pi$$

for some $n \in \mathbb{N}$.

Remark 1. Xie et al. (2019) show in Example 1, that $\hat{Q}^\pi$ suffers from the *curse of horizon* (Liu et al., 2018), which means that its variance increases exponentially as $H \to \infty$.

Another drawback of importance sampling is that the behavior policy is often not known in practice.

5.2 High Confidence Off-Policy Evaluation

One way of quantifying the risk of following a certain policy is to lower bound the expected return of this policy. Thomas et al. (2015) present a method to find such a lower bound with high probability by introducing a new concentration inequality. They call their approach High Confidence Off-Policy Evaluation (HCOPE).

Let $\pi_\theta(.|.)$ denote a policy with parameters $\theta \in \mathbb{R}^{n_\theta}$, where n_θ is the dimension of the parameter space of the policy. Furthermore, assume that the reward at each timestep t is in the interval $[r_-, r_+]$. For a trajectory $\tau = \{s_0, a_0, r_0, \dots, s_T, a_T, r_T\}$ and discount factor $\gamma \in [0,1]$, we set the normalized and discounted return to be

$$R(\tau) := \frac{\left(\sum_{t=0}^{T}\gamma^t r_t\right) - R_-}{R_+ - R_-} \in [0,1].$$

Choices for the lower and upper bounds could be $R_- = r_-(1-\gamma^{T+1})/(1-\gamma)$ and $R_+ = r_+(1-\gamma^{T+1})/(1-\gamma)$, if no sharper bounds can be obtained from domain-specific knowledge. For a trajectory τ, generated by a policy π_θ, we denote the performance of the policy as

$$\omega(\theta) := \mathbb{E}R(\tau).$$

We assume that a dataset $\mathcal{D}$, consisting of n trajectories $(\tau_i)_{i=1}^n$ generated by the behavior policies $(\pi_{\theta_i})_{i=1}^n$ with length at most T and corresponding labels, i.e.,

$$\mathcal{D} := \{(\tau_i, \theta_i) \mid i \in \{1, \dots, n\}, \tau_i \text{ generated by } \pi_{\theta_i}\}$$

is given. Denoting the evaluation policy with corresponding parameters θ as π_θ, importance sampling is used to compute an estimate $\hat{\omega}(\theta, \tau, \theta_i)$ of the performance $\omega(\theta)$ of the evaluation policy. Where $(\tau, \theta_i) \in \mathcal{D}$ and the estimate is given by

$$\hat{\omega}(\theta, \tau, \theta_i) := R(\tau) \prod_{t=0}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_i}(a_t|s_t)}. \tag{32}$$

The denominators in equation (32) are always strictly greater than zero, because a_t would not be an element of the trajectory τ, if $\pi_{\theta_i}(a_t|s_t) = 0$. Nevertheless, the authors use Assumption 1 for simplicity, but show that the lower bound also holds without this assumption.

As already mentioned in Remark 1, importance sampling can lead to large variances. Therefore, the authors modify an empirical version of Bernstein's inequality, which was proposed by Maurer and Pontil (2009). More precisely, they collapse the distribution of the random variables and normalize them to apply Maurer and Pontil's empirical Bernstein inequality. The result is of the following form.

Theorem 4 (Thomas et al. (2015))**.** Let $X_1, \dots, X_n$ be n independent and bounded real-valued random variables which satisfy $P(X_i \geq 0) = 1$ and $\mathbb{E}X_i \leq \mu$ for all $i \in \{1, \dots, n\}$. Then it holds for $c_1, \dots, c_n \in \mathbb{R}$, $Y_i := \min\{X_i, c_i\}$ for all $i \in \{1, \dots, n\}$ and $\delta > 0$ that

$$\begin{aligned} \mu \geq & \left(\sum_{i=1}^n \frac{1}{c_i}\right)^{-1} \sum_{i=1}^n \frac{Y_i}{c_i} - \left(\sum_{i=1}^n \frac{1}{c_i}\right)^{-1} \frac{7n\ln(2/\delta)}{3(n-1)} \\ & - \left(\sum_{i=1}^n \frac{1}{c_i}\right)^{-1} \sqrt{\frac{\ln(2/\delta)}{n-1} \sum_{i,j=1}^n \left(\frac{Y_i}{c_i} - \frac{Y_j}{c_j}\right)^2} \end{aligned} \tag{33}$$

with probability $1 - \delta$.

In the context of the discussed off-policy evaluation setting, given a threshold $\delta > 0$, this means that one can lower bound $\omega(\theta) =: \mu$ by the right hand side of inequality (33) with probability $1 - \delta$.

Vice versa, if one is given a lower bound, it is also possible to compute the probability that $\omega(\theta) \geq \mu_-$ for some $\mu_- \geq 0$. More precisely, the probability of $\omega(\theta) \geq \mu_-$ is

$$1 - \delta = \begin{cases} 1 - \min\left\{1, 2\exp\left(-\zeta^2\right)\right\}, & \text{if } \zeta \text{ is real and positive ,} \\ 0, & \text{otherwise,} \end{cases}$$

for

$$\zeta := \frac{-k_2 + \sqrt{k_2^2 - 4k_1k_3}}{2k_1}, \quad k_1 := \frac{7n}{3(n-1)},$$

$$k_2 := \sqrt{\frac{2}{(n-1)}\left(n\sum_{i=1}^{n}\left(\frac{Y_i}{c_i}\right)^2 - \left(\sum_{i=1}^{n}\frac{Y_i}{c_i}\right)^2\right)}, \quad k_3 := \mu_- \sum_{i=1}^{n}\frac{1}{c_i} - \sum_{i=1}^{n}\frac{Y_i}{c_i}.$$

One can see in (33) that the choice of the c_i is important to produce a tight lower bound. For large c_i the truncation is very moderate and the range of the Y_i is very large, vice versa the expected values of the Y_i get very small and also produce a loose lower bound. In order to deal with this trade-off, the authors propose to partition the dataset $\mathcal{D}$ into two subsets and set $c_i = c \in \mathbb{R}$ for all $i \in \{1, \ldots, n\}$. The first subset is used to predict the sample mean and sample variance of the lower bound, i.e., the right hand side of inequality (33). Using these predictions one can maximize the lower bound with respect to c. Then Theorem 4 can be applied with this maximizing c to obtain a lower bound.

5.3 Distributional Off-Policy Evaluation

Similar to online policy evaluation and control, especially in safety-critical applications it can be beneficial to work with the distribution of the return η_π as opposed to solely the expected value of the return. Wu et al. (2023) propose an algorithm which sequentially conducts Maximum Likelihood Estimation (MLE) to evaluate the distribution η_π of the return of an evaluation policy π and call it "Fitted Likelihood Estimation" (FLE). More precisely, one algorithm for finite horizon and one algorithm for infinite horizon MDPs is proposed respectively.

In both settings, it is assumed that a dataset of size n

$$\mathcal{D} := \{(s_i, a_i, r_i, s_{i+1}) | i = 1, \ldots, n\},$$

was generated from to the behavior policy π_b, . The elements are i.i.d. quadruples

$$(s_i, a_i, r_i, s_{i+1}),$$

where s_i is a state at some time t, $a_i \sim \pi_b(.|s_i), r_i = R(s_i, a_i)$ and $s_{i+1} \sim P(.|s_i, a_i)$.

For both algorithms to work, a function f is required that can generate samples $z \sim f(.|s, a)$ for any given state-action pair (s, a). Furthermore, the conditional likelihood $f(z|s, a)$ needs to be computable for any triple (z, s, a). Discrete histogram-based models, Gaussian mixture models, flow models and diffusion models are only some examples of function approximators that satisfy these requirements.

5.3.1 Finite Horizon Fitted Likelihood Estimation

We introduce the FLE algorithm for finite horizon MDPs, followed by corresponding convergence results.

Algorithm

Let $\eta_\pi^h(s,a)$ denote the distribution of the return under the policy π, starting at state-action pair $(s,a) = (s_h, a_h)$ at timestep h. Notice that

$$\eta_\pi = \mathbb{E}_{s\sim P_0(.), a\sim\pi(s)} \eta_\pi^1(s,a). \tag{34}$$

The dataset $\mathcal{D}$ is randomly and evenly split into H subsets $\mathcal{D}_1, \ldots, \mathcal{D}_H$. Where the number of subsets does not necessarily need to be H. This choice was made by the authors to simplify the analysis later on. Furthermore, we denote $\mathcal{F}_1, \ldots, \mathcal{F}_H$ as function classes that contain state-action conditional distributions.

Given these subsets of the dataset $(D_j)_{j=1}^H$ and the function classes $(\mathcal{F}_j)_{j=1}^H$, the algorithm starts at H and iterates until $h = 1$. The goal is to fit the target $\mathcal{T}^\pi \hat{f}_{h+1}$ at each timestep h, using $\hat{f}_{h+1}$ obtained from the previous iteration. Where $\mathcal{T}^\pi$ is the distributional Bellman operator (7) for the policy π, with a discount factor of $\gamma = 1$. As long as samples can be drawn from $\hat{f}_{h+1}(.|s,a)$ for any state-action pair (s,a), one can generate samples from $\mathcal{T}^\pi \hat{f}_{h+1}$, in order to learn the target. These samples are then used to perform MLE for fitting $\hat{f}_h$, which is used to estimate $\mathcal{T}^\pi \hat{f}_{h+1}$. After the last iteration, $\hat{f}_1$ is returned and approximates η_π^1. Utilizing (34), one obtains an estimate of η_π. Pseudo-code is given in Algorithm 6.

Algorithm 6 Fitted Likelihood Estimation – finite horizon

Input: Dataset $\{\mathcal{D}_h\}_{h=1}^H$ and function classes $\{\mathcal{F}_h\}_{h=1}^H$
Output: $\hat{f}_1 \approx Z_1^\pi$

for $h = H, \ldots, 1$ **do**
  $\mathcal{D}'_h = \emptyset$
  for $(s,a,r,s') \in \mathcal{D}_h$ **do**
    if $h < H$ **then**
      $a' \sim \pi(s')$
      $y \sim \hat{f}_{h+1}(\cdot \mid s', a')$
      $z \leftarrow r + y$
    else
      $z \leftarrow r$
    end if
    $\mathcal{D}'_h = \mathcal{D}'_h \cup \{(s,a,z)\}$
  end for
  $\hat{f}_h = \arg\max_{f\in\mathcal{F}_h} \sum_{(s,a,z)\in\mathcal{D}'_h} \log f(z \mid s,a)$
end for

Convergence Analysis

Let d_h^π be the state-action distribution induced by policy π at timestep h and set

$$d^\pi := H^{-1} \sum_{h=1}^H d_h^\pi. \tag{35}$$

The predictive error guarantees are given in terms of the *total variation distance*, which we define in the following.

Definition 12. For two distributions P_1 and P_2 on a set B, the *total variation distance* is defined as

$$d_{tv}(P_1, P_2) := \frac{1}{2}\|P_1 - P_2\|_1 = \frac{1}{2}\int_B |P_1(x) - P_2(x)|dx.$$

As is typical for OPE, a certain type of coverage assumption must be made.

Assumption 2 (Coverage)**.** For each timestep $h \in \{1, \ldots, H\}$, there exists a constant C, such that

$$\sup_{\substack{f_h \in \mathcal{F}_h \\ f_{h+1} \in \mathcal{F}_{h+1}}} \frac{\mathbb{E}_{s,a\sim d_h^\pi} d_{tv}^2 \left(f_h(s,a), \left[\mathcal{T}^\pi f_{h+1}\right](s,a)\right)}{\mathbb{E}_{s,a\sim \pi_b} d_{tv}^2 \left(f_h(s,a), \left[\mathcal{T}^\pi f_{h+1}\right](s,a)\right)} \leq C \ .$$

Let $\Delta(B)$ denote the set of all distributions over a set B. Furthermore, let $\hat{f}_1, \ldots, \hat{f}_H : \mathcal{S} \times \mathcal{A} \mapsto \Delta([0, H])$ be a sequence of functions and assume there exist $\zeta_1, \ldots, \zeta_H \in \mathbb{R}$ such that

$$\left(\mathbb{E}_{s,a\sim\pi_b} d_{tv}^2 \left(\hat{f}_h(s,a), \left[\mathcal{T}^\pi \hat{f}_{h+1}\right](s,a)\right)\right)^{1/2} \leq \zeta_h, \forall h \in \{1, \ldots, H\}. \tag{36}$$

Combining inequality (36) with Assumption 2 and denoting $\hat{f} := \mathbb{E}_{s\sim P_0(.), a\sim\pi(s)} \hat{f}_1(s,a)$, the authors show that

$$d_{tv}(\hat{f}, \eta_\pi) \leq \sqrt{C} \sum_{h=1}^{H} \zeta_h.$$

This means that small supervised learning errors imply small prediction error of the previously described OPE Algorithm 6, given that Assumption 2 holds.

To answer the question of which form the $\zeta_.$ in inequality (36) are, another assumption needs to be made.

Assumption 3 (Bellman completeness)**.**

$$\max_{h\in[H], f\in\mathcal{F}_{h+1}} \min_{g\in\mathcal{F}_h} \mathbb{E}_{x,a\sim\pi_b} d_{tv} \left(g(x,a), \left[\mathcal{T}^\pi f\right](x,a)\right) = 0 \tag{37}$$

This means that $\mathcal{T}^\pi \hat{f}_{h+1} \in \mathcal{F}_h$ at each iteration of Algorithm 6. Furthermore, we need the following definition.

Definition 13. Let $\mathcal{F}$ be a function class whose elements map from B to $\mathbb{R}$. The *bracket* $[l, u]$ of two functions is defined as

$$\{f \mid f \in \mathcal{F},\ \forall x \in B :\ l(x) \leq f(x) \leq u(x)\}.$$

An ε bracket is a bracket $[l, u]$, satisfying $\|l - u\| \leq \varepsilon$. The minimum number of ε-brackets needed to cover $\mathcal{F}$ is denoted as $N_{[]}(\varepsilon, \mathcal{F}, \|.\|)$ and called the *bracketing number* of $\mathcal{F}$ with respect to the metric $\|.\|$.

Under Assumption 3, the authors show that inequality (36) holds for

$$\zeta_h = \begin{cases} \left(\frac{4H}{n} \log(|\mathcal{F}_h| \; H/\delta)\right)^{1/2}, & \text{if } |\mathcal{F}_h| < \infty, \\ \left(\frac{10H}{n} \log\left(N_{[]}((nH)^{-1}, \mathcal{F}, \|.\|_\infty) \; H/\delta\right)\right)^{1/2}, & \text{else.} \end{cases}$$

with probability at least $1-\delta$ for all $h \in \{1, \dots, H\}$.

Summing everything up, one obtains the following result.

Theorem 5 (Wu et al. (2023))**.** If inequality (36), Assumption 2 and Assumption 3 hold, the error in Algorithm 6 can be bound by

$$d_{tv}(\hat{f}, \eta_\pi) \leq \begin{cases} \sqrt{C} \sum_{h=1}^{H} \sqrt{\frac{4H}{n} \log\left(|\mathcal{F}_h| \; H/\delta\right)}, & \text{if } |\mathcal{F}_h| < \infty \;\; \forall h \in \{1, \dots, H\}, \\ \sqrt{C} \sum_{h=1}^{H} \sqrt{\frac{10H}{n} \log\left(N_{[]}((nH)^{-1}, \mathcal{F}, \|.\|_\infty) \; H/\delta\right)}, & \text{else.} \end{cases}$$

5.3.2 Infinite Horizon Fitted Likelihood Estimation

Again, we present the FLE algorithm for infinite horizon MDPs first and show the corresponding convergence results afterwards.

Algorithm

In the case of infinite horizon MDPs, the discount factor γ is in the interval $(0, 1)$. Furthermore, we assume without loss of generality that the rewards are in the interval $[0, 1]$, which implies that $Z^\pi(.,.) \leq \frac{1}{1-\gamma}$ by (3). We denote the function class that is input to the algorithm as $\mathcal{F} \subset \mathcal{S} \times \mathcal{A} \mapsto \Delta([0, \frac{1}{1-\gamma}])$. The dataset $\mathcal{D}$ is evenly and randomly split into T subsets, as in the finite horizon case. The algorithm starts at $t = 1$ and at each timestep t, the target distribution to fit by MLE is $\mathcal{T}^\pi \hat{f}_{t-1}$. After the last iteration, η_π can be approximated by

$$\mathbb{E}_{s \sim P_0(.), a \sim \pi(s)} \hat{f}_t(s, a) \approx \eta_\pi.$$

The corresponding pseudo-code is given in Algorithm 7.

Algorithm 7 Fitted Likelihood Estimation – infinite horizon

Input: Dataset $\{\mathcal{D}_t\}_{t=1}^T$ and function class $\mathcal{F}$
Output: $\hat{f}_T$

for $t = 1, \dots, T$ **do**
  $\mathcal{D}'_t = \emptyset$
  for $(s, a, r, s') \in \mathcal{D}_t$ **do**
    $a' \sim \pi(s')$
    $y \sim \hat{f}_{t-1}(\cdot \mid s', a')$
    $z \leftarrow r + \gamma y$
    $\mathcal{D}'_t = \mathcal{D}'_t \cup \{(s, a, z)\}$
  end for
  $\hat{f}_t = \arg\max_{f \in \mathcal{F}} \sum_{(s,a,z) \in \mathcal{D}'_t} \log f(z \mid s, a)$
end for

Convergence Analysis

Analogously to (35), we denote the state-action distribution of a given policy π as $d^\pi := (1-\gamma)^{-1} \sum_{h=1}^{\infty} d_h^\pi$. Chung and Sobel (1987) have shown that the distributional Bellman operator $\mathcal{T}^\pi$ for $\gamma < 1$ is no contraction mapping in total variation distance. Therefore, Wu et al. (2023) show that $\mathcal{T}^\pi$ is a contraction mapping under the *average p-Wasserstein distance*. For two distributions μ and ν, the latter is given by

$$\left(\mathbb{E}_{s,a\sim d^\pi}[W_p(\mu(s,a),\nu(s,a)]^{2p})\right)^{\frac{1}{2p}}.$$

Furthermore, the authors show that

$$W_p^p(\mu,\nu) \leq \left(\sup_{x,y\in\mathcal{S}} \|x-y\|\right)^p \, d_{tv}(\mu,\nu),$$

which allows to transfer the results for the estimation error of MLE under the total variation distance, to the Wasserstein distance. Otherwise, the procedure is quite similar to the finite horizon MDP setting and we omit it for brevity. The final result for Algorithm 7 is the following.

Theorem 6. Under assumptions which are analogous to Assumption 2 and Assumption 3, setting $\hat{f} := \mathbb{E}_{s\sim P_0, a\sim\pi(s)} \hat{f}_T(s,a)$ and

$$\iota := \begin{cases} \log(|\mathcal{F}|/\delta), & \text{if } |\mathcal{F}| < \infty, \\ \log\left(N_{[]}\left(\frac{1-\gamma}{n}, \mathcal{F}, \|\cdot\|_\infty\right)/\delta\right), & \text{otherwise,} \end{cases}$$

and picking

$$T = \log\left(C^{\frac{1}{2p}} \, \iota^{\frac{1}{2p}} \left(1-\gamma^{\frac{1}{2}}\right)^{-1} n^{-\frac{1}{2p}}\right) / \log\left(\gamma^{1-\frac{1}{2p}}\right),$$

the error in Algorithm 7 can be bound by

$$d_{w,p}\left(\hat{f}, \eta_\pi\right) \leq \widetilde{O}\left(\frac{C^{\frac{1}{2p}} \, \iota^{\frac{1}{2p}}}{(1-\gamma)^{\frac{5}{2}}} \, n^{-\frac{1}{2p}}\right)$$

with probability at least $1-\delta$.

6 Environments

As the number of solar and wind power plants around the world grows (Chen and Ji, 2024) and because these sources of energy are volatile due to their dependence on sunlight and wind, the issue of storing electricity at times of high production is becoming increasingly important. A frequently used method are pumped hydro storage systems. During periods of high power generation, water from a lower lying reservoir is pumped to a higher lying reservoir. Vice versa, the water is turbinated at periods of low power generation but high demand.

In the following, we will describe the two environments in which we apply the RL algorithms. Both environments are simulation models of a reversible pump turbine implemented in the simulation software *Simulink* (Documentation, 2020). The models were developed by colleagues from the "Institute of Energy Systems and Thermodynamics" at TU Wien as part of the "RELY – Reliable Reinforcement Learning for Sustainable Energy Systems" project and other previous work (Tubeuf et al., 2023).

In both models, the goal of the agent is to efficiently control the pump turbine during a specific operation sequence. A sketch of the pump turbine is shown in Figure 6.1.

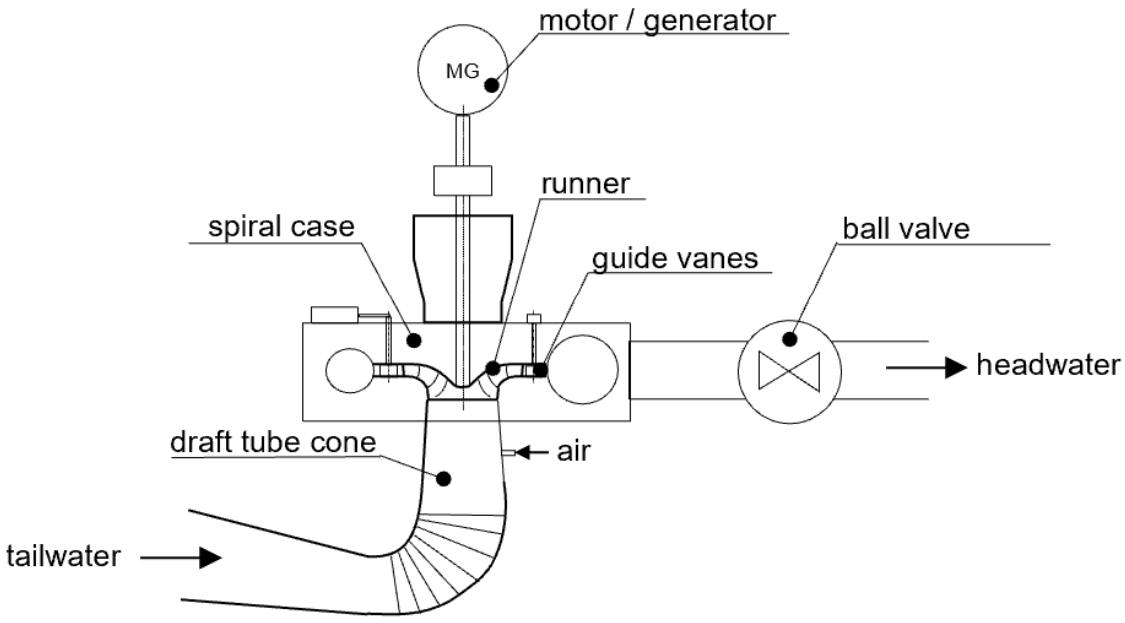


Figure 6.1: Sketch of the pump turbine, which represents the RL environments. (Tubeuf et al., 2024)

The pump turbine can operate in pump and turbine mode. Increasing the efficiency can lead to faster switching between these modes, resulting in economic benefits and the

possibility to balance out more fluctuations in the electricity grid stemming from volatile energy sources.

While Simulink is a widely used software for building complex physical simulation models, the RL community writes most of the codebase in the *Python* programming language. In principle, there are multiple options to bridge this gap. However, the number of choices is reduced by the variable step size solvers on which our Simulink models are based and by other requirements. The solution we use is a Python package (Brust, 2025), which runs the simulation in Simulink and transfers input and output data between the simulation model and the RL algorithms, which are written in Python, via TCP/IP communication.

6.1 Blow-Out Model

For pump start-up it is required to blow out the runner. This means that air from a compressor is blown into the runner to replace the water therein by air. Doing so results in less torque and therefore a decreased consumption of electric power and less wear and tear, because air causes less friction than water.

The action space $\mathcal{A}$ is binary, with the actions corresponding to blowing air into the runner or not. The state space $\mathcal{S}$ is three dimensional with two dimensions being continuous and the third one being binary. One dimension corresponds to the water level in the runner, taking values in the interval $[0, w_{\max}]$ and measured in meters. The other continuous dimension corresponds to the total amount of air blown into the runner. Which action was taken on the timestep before the current timestep, is encoded in the binary state. In total, a state at time t is of the form $s_t = (s_{\text{water level}}, s_{\text{air mass}}, s_{\text{previous action}})_t$.

The reward function is given as

$$R(s_t, a_t) = w_1 \; r_{\text{water level}} + w_2 \; r_{\text{air}} + w_3 \; r_{\text{previous action}},$$

where

$$r_{\text{water level}} = \begin{cases} -1, & \text{if } s_{\text{water level}} < l_{\min}, \\ 1, & \text{if } l_{\min} \leq s_{\text{water level}} \leq l_{\max}, \\ -s_{\text{water level}}, & \text{if } l_{\max} < s_{\text{water level}} \end{cases}$$

is supposed to encode the need of keeping the runner in a blown-out state, i.e., a water level below $l_{\max}$ meters. Furthermore, the water level must be above a critical level of $l_{\min}$ meters to prevent air dissipating into the tailwater. The costs of using the compressor to blow air into the runner are reflected by $r_{\text{air}} = -a_{\text{air}}$. Lastly, $r_{\text{previous action}} = -\mathbb{1}_{|a_{t,air} - a_{t-1,air}|=1}$ reflects the costs of wear and tear caused by frequently switching the action. Summing it up, the goal is to keep the runner in a blown-out state for as long as possible, while minimizing the amount of air blown in and minimizing the number of action changes.

Setting the weights to $w_1 = 1, w_2 = 0.2, w_3 = 0.8$, empirically turned out to lead to good performance. Each simulation has a duration of 120 seconds and one step is taken every second.

6.2 Simulation of a Commercially used Pump Turbine

The second environment is also a Simulink (Documentation, 2020) simulation model of a pump turbine. While the simulation model aims to represent a pump turbine located

at a laboratory of TU Wien, the pump in the laboratory is modified to represent a real, commercially used pump turbine which is part of a pumped hydro storage system in Kühtai, Austria. This means that the states can also be measured in a real-world setting and the actions can also be controlled in a real-world setting. While the goal for the environment described in Section 6.1 was to only control the blow-out process, the second use case is aiming to control the whole pump start-up process. This means that the pump turbine is controlled from stand-still to synchronized operation by means of a steady flow rate.

The action space $\mathcal{A}$ consists of four actions. The "air valve" action is binary and controls whether air is blown into the runner or not. The "guide vanes" action controls the guide vane opening and can open/close the guide vane incrementally or keep it in its current position. The "speed switch" action decides when the runner starts rotating and is binary. Lastly, the "ball valve" action is binary and opens or closes the ball valve, which sits between the pump turbine and the headwater reservoir.

The state space $\mathcal{S}$ consists of 19 states which are characterized in Table 6.1. The ball valve can be opened and closed continuously and controls how much water can flow. The same holds for the guide vane opening. Whether the draft tube can be considered as blown out or not is signalled by the "water level switch max."

Very roughly speaking, the control of the pump turbine works as follows. Air is blown into the runner to displace the water therein by air. During this sequence, the ball valve is usually closed because the air would dissipate otherwise. If the runner starts rotating and there is a lot of water inside, a high torque acts on the machinery and causes heavy wear and tear. However, blowing in air costs time and if too much air is blown in, i.e., more than can be hold in the storage air vessels, costs for not being able to run the pump turbine until the air vessels are again filled up with compressed air arise. So the agent gets a reward which equals these costs.

Increasing rotational speed of the runner leads to a higher flow rate of the water. Once the rotational speed equals the synchronizational speed, the synchronizational speed is set to be kept constant and the torque state is set to zero. This artificial adjustment doesn't aim to represent physical laws properly, but helps to learn a good policy, because then the reward corresponding to torque isn't negative anymore. This has no impact on the simulation. After the machine unit is synchronized with the electricity grid, the ball valve and the guide vanes are being opened until a desired flow rate is set in. Once the rotational speed reached the target speed, the flow rate equals the target flow rate for 15 consecutive seconds and the ball valve is open, the simulation is stopped and the environment is reset.

Denoting the last simulation time the reward was calculated for as t_{old}, the reward function at time t is of the form

$$\begin{aligned} R(t) =& (t - t_{\text{old}})[\text{f\_r\_r}(t) - w_1 \text{ torque(t) rot\_speed}(t) \\ & - w_2 - w_3 \text{ air\_mass}(t)] - w_4 \text{ ball\_valve switch}(t) \\ & - w_5 \text{ air\_valve\_switch}(t) - w_6 \text{ guide\_vane\_action}(t) - \text{air\_volume}(t) \\ & - \text{shutdown}(t) + \text{timeout}. \end{aligned}$$

Where flow rate reward is given as

$$\text{f\_r\_r}(t) = \begin{cases} w_7 |\text{target\_flow\_rate} - \text{flow\_rate}(t)|, & \text{if } |\text{target\_flow\_rate} - \text{flow\_rate}(t)| \leq 0.001, \\ -w_8 \text{ flow\_rate}(t) \text{ height}(t), & \text{otherwise} \end{cases}$$

and $w_1, \ldots, w_8$ are the weights for the corresponding costs. In our case, $t - t_{\text{old}}$ is constantly 0.2, since new states are input to the simulation every 0.2 seconds.

The costs resulting from not operating at the target flow rate are encoded in the reward function. One can also see the negative impact of high torque at high rotational speed. The constant negative reward reflects opportunity costs resulting from slowly reaching operational mode. Furthermore, the costs resulting from blowing in air are also part of the reward function. Switching the ball valve, air valve and guide vane action very frequently leads to higher wear and tear and therefore leads to a negative reward.

As already mentioned, a one time reward is given if air volume is above a critical value. Furthermore, blowing in too much air results in a water level in the draft tube which is below a critical value. This is signalled by "water level switch min." In this case, the episode/simulation is terminated and the agent receives a high negative reward. Another highly negative reward is given, if the head water pressure gets too high. Then the simulation is also terminated. These two rewards are given by shutdown(t) in the reward function. A maximum episode length of 150 seconds is used. Empirical evidence has demonstrated that values within this range yield the best training outcomes. If an episode is not terminated before this time the agent receives a reward of timeout $= -150 * w_9$.

Experts in the field of pumped storage power plants and mechanical engineering were conducted to determine the opportunity and material costs appearing in the reward function. Manual control, based on expert knowledge, led to a reward of -0.0625 and took about 83 seconds. We will use these values as a baseline for our RL algorithms later on.

State	**Value Range**	**Unit**	**Description**
ball valve position	$[0, 1]$	—	position of ball valve [closed, open]
rotational speed	$[0, \infty)$	rpm	rotational speed of the runner
torque	$[0, \infty)$	Nm	—
synced	$\{0, 1\}$	—	rot. speed below target, or synch. speed reached
flow rate	$[0, \infty)$	m^3/s	flow rate of the water
flow rate target	$\{0, 1\}$	—	flow rate below/above target, or reached
guide vane opening	$[0, a_{\max}]$	m	opening of guide vane [closed, open]
water level switch max	$\{0, 1\}$	—	draft tube is considered blown out, or not
air mass	$[0, \infty)$	kg/s	current mass of air blown in per second
air valve position	$[0, 1]$	—	position of air valve
time	$[0, 150]$	s	simulation time
water level switch min	$\{0, 1\}$	—	water in draft tube is below or above a min. level
air volume	$[0, V_{\max}]$	kg	total amount of air blown in
max pressure	$\{0, 1\}$	—	pressure at critical level (no/yes)
air valve switch	$\{0, 1\}$	—	air valve action switches from $t-1$ to t
height	$[0, h_{\max}]$	m	height diff. between water source and destination
episode done	$\{0, 1\}$	—	flow rate at target for 15 consecutive sec. (no, yes)
ball valve switch	$\{0, 1\}$	—	ball valve action switches from $t-1$ to t
guide vane action	$\{0, 1\}$	—	guide vane action is 0 or not

Table 6.1: All 19 state dimensions in the simulation model.

7 Results and Discussion

In the following, we present the learning results for the environments described in Chapter 6.

Section 7.1 deals with the results for the environment described in Section 6.1. We describe the search for DistrRL algorithm hyper parameters, yielding stable results. Furthermore, we compare the performance of the algorithms presented in Chapter 6 and conclude that an optimal policy has been found for the environment. In Section 7.2, we present some ideas on how to leverage the distributions learned by the DistrRL algorithms to score the reliability of the policies found. Section 7.3 is about the results for the more complex environment described in Section 6.2. We analyze a found policy and discuss possible measures to learn a policy that meets all objectives.

7.1 Learning an Optimal Blowout Policy

In this Section 7.1, we present the results obtained by the DistrRL algorithms introduced in Chapter 3. All agents were trained for 305,000 timesteps and the other hyper parameters can be seen in Table 7.1. We trained all algorithms on different sets of hyper parameters and chose the hyper parameter setting which led to the most stable convergence and highest online-policy evaluation result. This choice of hyper parameters was made for each algorithm separately which makes comparing the learning performances, especially at the beginning of the training, harder. On the other hand, one can argue that this method ensures that the best possible final performance, across the respective hyper parameter sets, is depicted in Figure 7.2.

The hyper parameters which empirically turned out to have the biggest impact on the learning results were the following:

- total number of training steps,
- learning rate(s) of optimizer(s) for neural network backpropagation,
- exploration fraction, i.e., fraction of total number of training steps where exploration rate decreases until final exploration rate,
- initial exploration rate, i.e., exploration rate at the start of training.

The training curves are depicted in Figure 7.2. On the $x-$axis one can see the number of training episodes, while the cumulative reward is displayed on the $y-$axis. The cumulative reward is the sum of rewards obtained during one episode, i.e., $\sum_{t=0}^{T} r_t$. We averaged the cumulative reward curves over six seeds. The solid lines correspond to the mean of the cumulative reward curves of the respective algorithm. The shaded areas around these lines correspond to the standard deviation. To increase readability of the plot, we only plot the cumulative reward of every 20th training episode. The dotted horizontal line corresponds to the highest cumulative reward obtained by the optimal policy.

C51

Looking at the cumulative reward curve of the C51 algorithm, we can see that the algorithm learns relatively fast compared to QR-DQN and IQN. One reason for that might be the higher exploration fraction of the latter two algorithms. A higher exploration fraction, leads to a slower decay of the probability of taking random actions.

OS-C51

The OS-C51 algorithm learns comparably fast at the beginning. From training episodes 1300 to about 1600, OS-C51 has a very low variance across the different seeds and outperforms C51. Nevertheless, performance also slightly decreases at the end, similar to C51 and IQN. Furthermore, notice that the number of atoms used in the OS-C51 algorithm is 4, which is much less than the number of atoms/quantiles used in the other algorithms. It is quite interesting that such good performance could be obtained by a comparably small number of atoms. On the other hand, a distribution approximated by only four atoms might be a drawback when it comes to computing risk measures from the obtained probability distributions.

QR-DQN

QR-DQN learns faster than IQN in the beginning. This might be noteworthy, as the exploration fraction hyper parameter for QR-DQN is 0.8, while the same hyper parameter is 0.6 for IQN. We can also see that the variance of QR-DQN is very low during the last 600 training episodes. Furthermore, it outperforms the other algorithms on these last 600 episodes.

IQN

The IQN algorithm takes comparably long to learn a good policy, but seems to slightly outperform C51 and OS-C51 in the end. Nevertheless, IQN has a slightly worse performance and higher variance than QR-DQN at the last training episodes.

FQF

Looking at Figure 7.2, one can see that we have not been successful in training the FQF algorithm. We tried out 68 different sets of hyper parameters, which is much more than the number of sets for the other algorithms (C51: 30, OS-C51:17, QR-DQN: 10, IQN: 9). Since the FQF algorithm trains a neural network for the quantile values and for the quantile fractions, it is the most expensive DistrRL algorithm we applied, from a computational perspective.

Figure 7.1 shows the training performance of the FQF implementation applied on the "PongNoFrameskip" environment[1]. The environment is part of the Arcade Learning Environment (Bellemare et al., 2013), which is frequently used to evaluate the performance of RL algorithms. Figure 7.1 indicates that the FQF implementation we use is working and we therefore assume that the performance on the pump environment might be heavily increased by further hyper parameter tuning.

[1] https://gymnasium.farama.org/v0.28.1/environments/atari/pong/

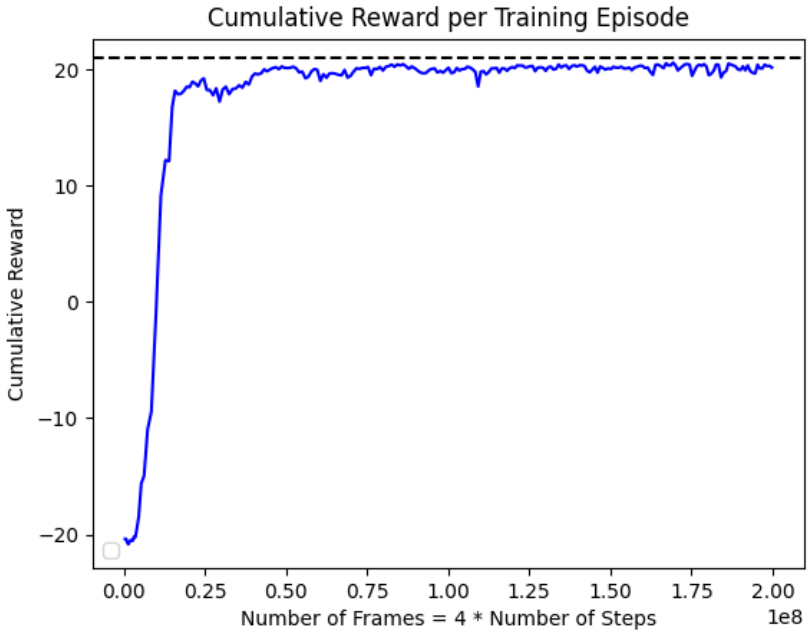


Figure 7.1: Summed reward per episode during training the FQF algorithm on the "PongNoFrameskip-v4" environment. The total number of training steps is 50,000,000, resulting in a total number of 200,000,000 frames. The dashed horizontal line indicates the maximum cumulative reward that can be obtained per episode.

Hyper Parameter	**C51**	**OS-C51**	**QR-DQN**	**IQN**	**FQF**
# of atoms/quantiles	500	4	100	100	128
learning rate	0.0004	0.0005	0.0005	0.0004	0.02
train frequency	8	8	8	8	8
batch size	1024	1024	1024	1024	1024
target update interval	10,000	10,000	10,000	10,000	10,000
discount factor γ	0.9995	0.9995	0.9995	0.9995	0.9995
replay buffer size	122,880	122,880	122,880	122,880	122,880
exploration fraction	0.4	0.4	0.8	0.6	0.5
initial exploration rate	0.4	0.4	0.4	0.4	0.6
final exploration rate	0	0	0	0	0
# of steps until network starts training	1	1	122,880	122,880	30,700
soft update coefficient			1	1	
# of gradient steps			1	1	
# of samples τ				64	
# of samples τ'				64	
fraction learning rate					1e-08
# of cosines				64	64

Table 7.1: Hyper parameter settings of DistrRL algorithms used in the experiments.

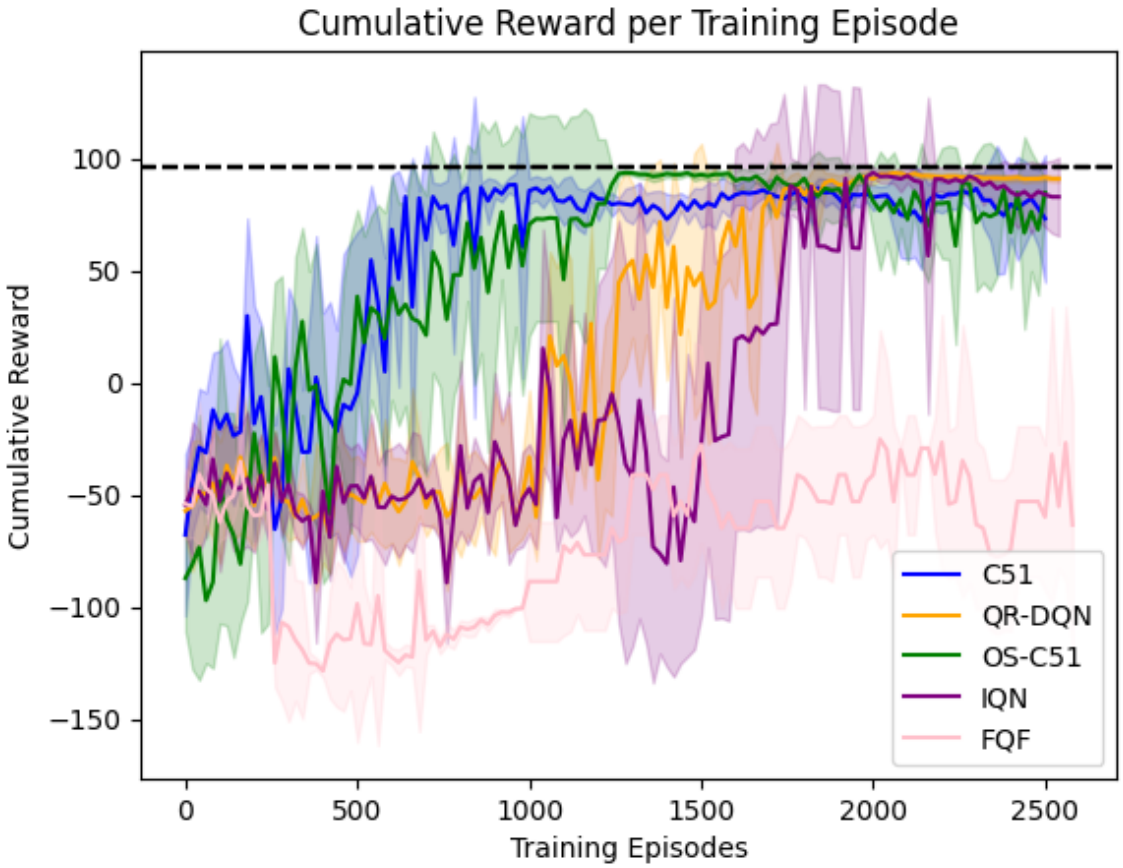


Figure 7.2: Summed rewards per training episode for different DistrRL algorithms. Training results are averaged over 6 seeds. Solid lines are the mean and shaded areas indicate the standard deviations over the seeds.

Optimal Policy

The policy which got the maximum online evaluation cumulative reward of about 96.3347 was learnt by QR-DQN. In Figure 7.3 the normalized water level in the runner, which results from this policy is depicted. A value of 1 corresponds to $w_{\max}$ meters and a value of 0 corresponds to a water level of 0 meters. The upper horizontal line indicates the threshold below which the runner is considered to be blown out. The lower horizontal line indicates the critical threshold below which the water level should not drop. The water level drops instantly at the beginning of the simulation. This indicates that air is blown in. Until about second 40 air is continuously blown into the runner, as the water level drops. Then no more air is blown in and the water level is rising again, due to leakage effects. This happens until the end of the simulation, where the water level is exactly below the threshold that marks that the runner is blown out. We can see that the minimum number of action switches is one, otherwise the water level would be outside the interval where the runner is considered to be blown out. Furthermore, the amount of air blown in is also minimized, since the water level is exactly below the upper threshold at the end of the episode. Lastly, we can see that the time interval in which the water level is above the

threshold is minimized, as the water level drops instantly after simulation start, because air is instantly blown in. Overall, we conclude that the learnt policy is optimal.

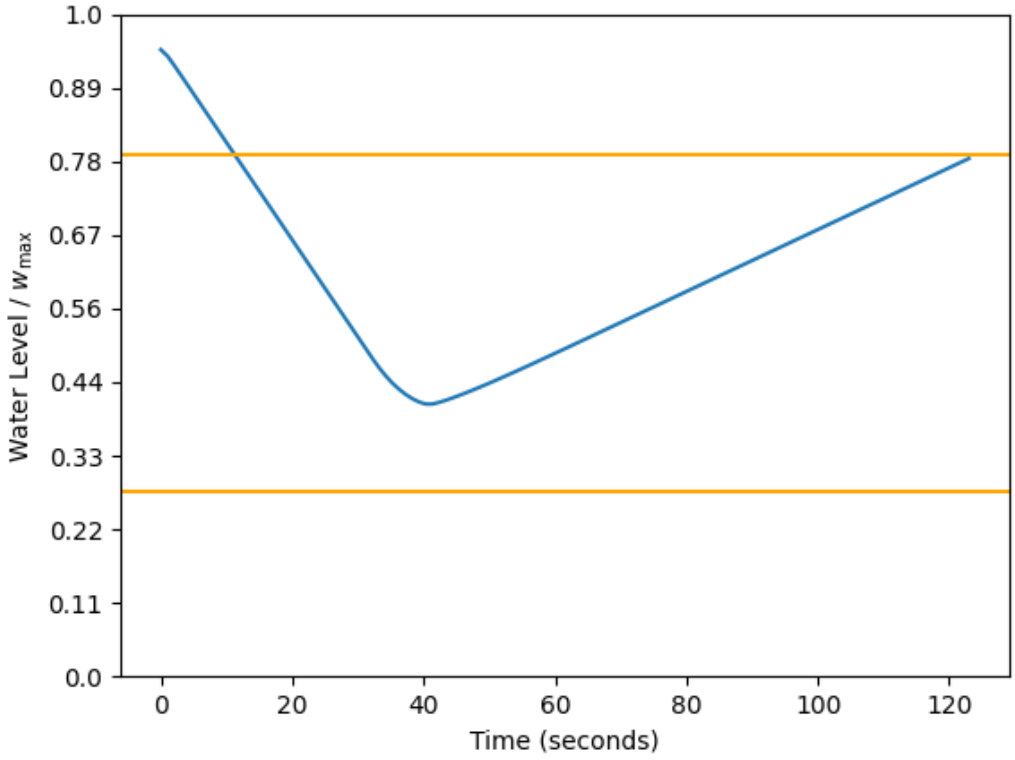


Figure 7.3: Normalized water level in the runner resulting from following the optimal policy.

7.2 Utilization of the Learnt Return Distributions

In this Subsection 7.2 we analyze the return distributions corresponding to the learnt optimal policy, which we described in Section 7.1. Since each return distribution depends on the starting state and action at time zero, we compare the distributions at different state-action pairs (s_0, a_0). We assume that the compressor always has full capacity in the beginning, which corresponds to $s_{\text{air mass}} = 0$ at the beginning.

The Wasserstein distance is used for the pairwise comparison of the distributions. First, we compare the distributions for $s_{\text{previous action}} = 1$, action $a = 1$ and varying values of the water level $s_{\text{water level}} \in [0, w_{\max}]$. Looking at Figure 7.4 we can see the pairwise similarity of the distributions for a different water level at the beginning. Note that we have normalized the water level to be in the interval $[0, 1]$. The plot indicates that starting at a normalized water level below 0.58 leads to similar return distributions, when the other state dimensions and the action are fixed.

The distributions corresponding to normalized water levels above 0.9 are quite different from the distributions of water levels below 0.7m. Intuitively, this makes sense since the

normalized target interval for the water level is $[0.28, 0.79]$, so the agent needs to follow a different policy depending on starting in the target interval or above. The different policy then leads to a different structure of the reward sequence obtained.

Two such distributions are shown in Figure 7.5. One distribution corresponds to a normalized initial water level of 1 and the other distribution's normalized initial water level is 0.43. These are approximately the water levels that lead to the largest Wasserstein distance, according to Figure 7.4. Notice, that the support of the return distributions is unequal to the support of the true return distribution. The reason for that is that the neural network used in QR-DQN is not properly scaled, because scaling has no effect on improving the policy.

The red dot in the top left corner indicates the deterministic start value of the water level, which we used during training the agents. Moving either horizontally or vertically from this point, one can see how much the return distribution changes, measured in Wasserstein distance.

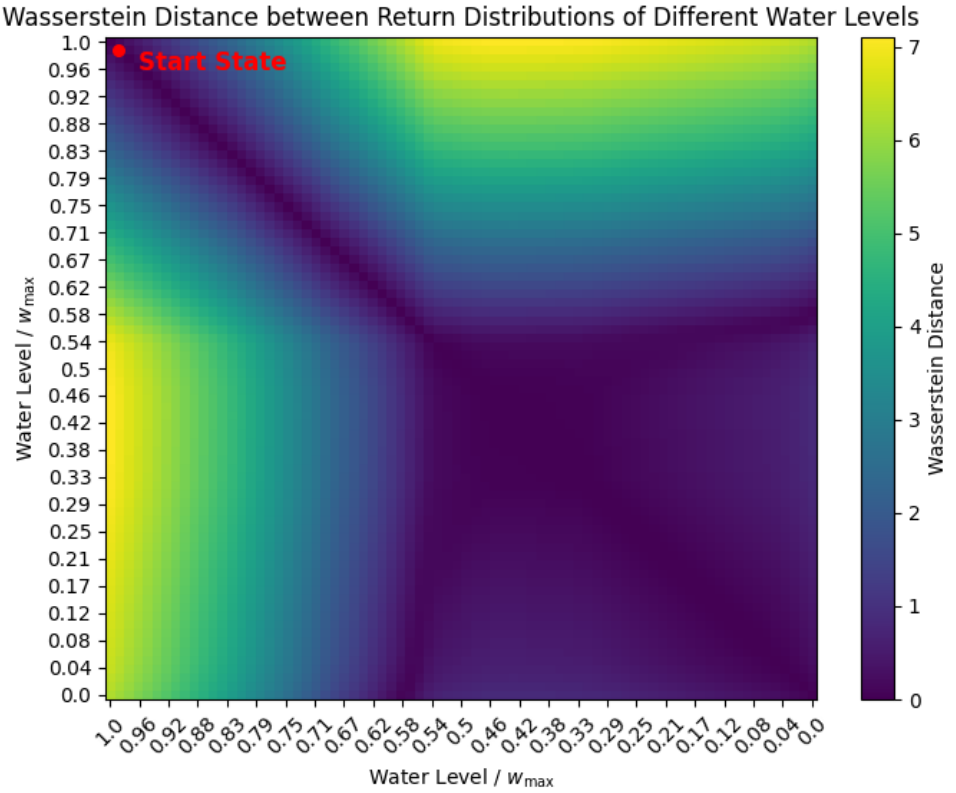


Figure 7.4: Pairwise Wasserstein distances of return distributions for varying normalized water levels and $s_{\text{previous action}} = 1$, $s_{\text{air mass}} = 0$, and action $a = 1$. The red dot indicates the start state used during training.

We also compare the similarity of the return distributions for varying actions $a \in \{0, 1\}$ and varying previous actions $s_{\text{previous action}} \in \{0, 1\}$. The results are very similar to the ones in the setting discussed above. For the sake of completeness, we have included the plots in Figure 8.1 in the Appendix.

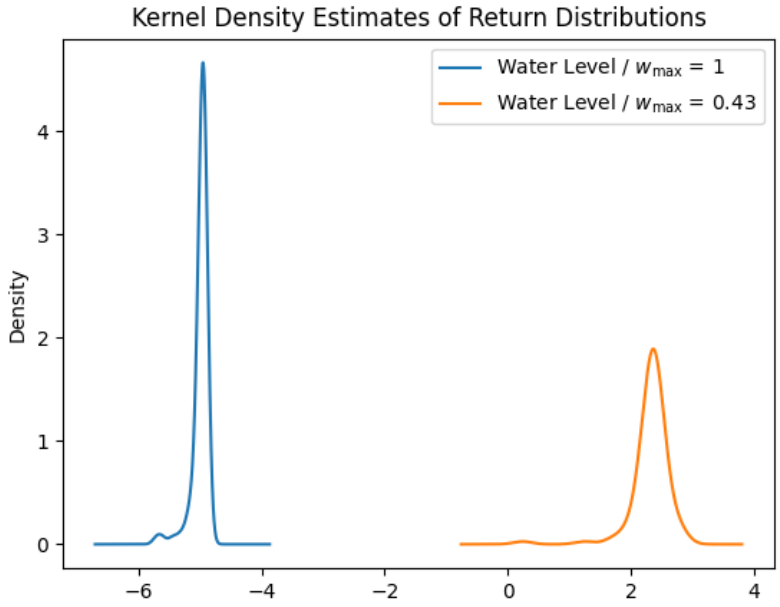


Figure 7.5: Comparison of two return distributions which are relatively different, according to the pairwise computed Wasserstein distances. The initial states of the distributions are $(s_{\text{water level}}/w_{\max}, s_{\text{previous action}}, s_{\text{air mass}}) = (0.43, 1, 0)$, $(s_{\text{water level}}, s_{\text{previous action}}, s_{\text{air mass}}) = (w_{\max}, 1, 0)$ and the action is $a = 1$.

Next, we calculate the variance of return distributions across different initial state-action pairs. Again, we fix $s_{\text{previous action}} = 1$ and action $a = 1$ at first and vary the water level in the interval $[0, w_{\max}]$. The variances are depicted in Figure 7.6. One can see that return distributions corresponding to an initial water level in the normalized interval $[0.5, 0.8]$ have a much higher variance. This indicates that starting from such water levels results in a relatively large variation in return. In other words, the policy has a relatively uncertain outcome when the episode starts with a water level in this interval.

Changing the initial previous action state and the action, leads to a similar variance structure across the initial water level. The corresponding plots are shown in Figure 8.2 in the Appendix.

7.3 DistrRL on a Simulation Model of a Commercially used Pump Turbine

Designing a reward function that represents real world costs and helps the agent to learn a good policy at the same time is not trivial. Providing a detailed reward function that leads the agent to learn a "good policy" means, in the extreme case, replicating a manual policy already used by practitioners. This is why we implemented a reward function, which solely reflects costs in the real world. The idea is that the agent will not be biased by existing heuristics and will find policies that are more efficient than those already known.

For the more complex environment, described in Section 6.2, we applied the C51 and

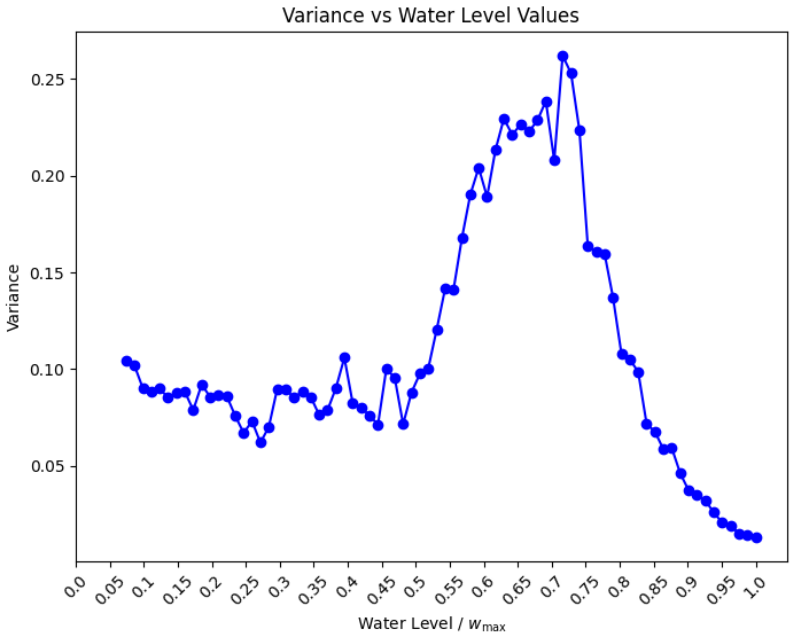


Figure 7.6: Variances of return distribution with initial state $(s_{\text{water level}}, s_{\text{previous action}}, s_{\text{air mass}}) = (s_{\text{water level}}, 1, 0)$ and action $a = 1$, where $s_{\text{water level}}$ varies in $[0, w_{\max}]$.

QR-DQN algorithms to find good policies. Similar as in Section 7.1, we did training runs on different sets of DistrRL algorithm hyper parameters. Even though, the search was done on broader ranges and a finer grid for each hyper parameter than in Section 7.1, the results were much less stable in terms of variance of summed episode rewards during training across different seeds. Furthermore, we were not able to reliably find a policy that meets all targets. A training run that is quite representative of the problems we faced is depicted in Figure 7.7. Training episodes are shown on the $x-$axis and summed rewards per episode are given on the $y-$axis. The dark blue line is the moving average over the last 50 values and only every $5-$th data point is plotted to increase readability. At the end of training, the graph seems to converge. Nevertheless, the corresponding policy does not meet all targets. In the following, we discuss the learnt policy.

The agent learns to increase the rotational speed until the target level relatively fast, as can be seen in Figure 7.8. The amount of air blown in during this time is rather low and therefore the torque increases fast. This can be seen in Figure 7.9. Furthermore, one can see that no more air is blown in shortly after the target rotation speed is met. This makes sense since blowing in more air after the rotation speed is at the target level only introduces additional costs.

The flow rate is controlled by opening/closing the ball valve and guide vanes. These states can be seen in Figure 7.10. The ball valve is fully open from a simulation time of approximately 53 seconds until the end of the episode. So the target of the ball valve being open at the end of the simulation is met. Moreover, the guide vanes are open from a simulation time of approximately 27 seconds until the end. This is problematic since it

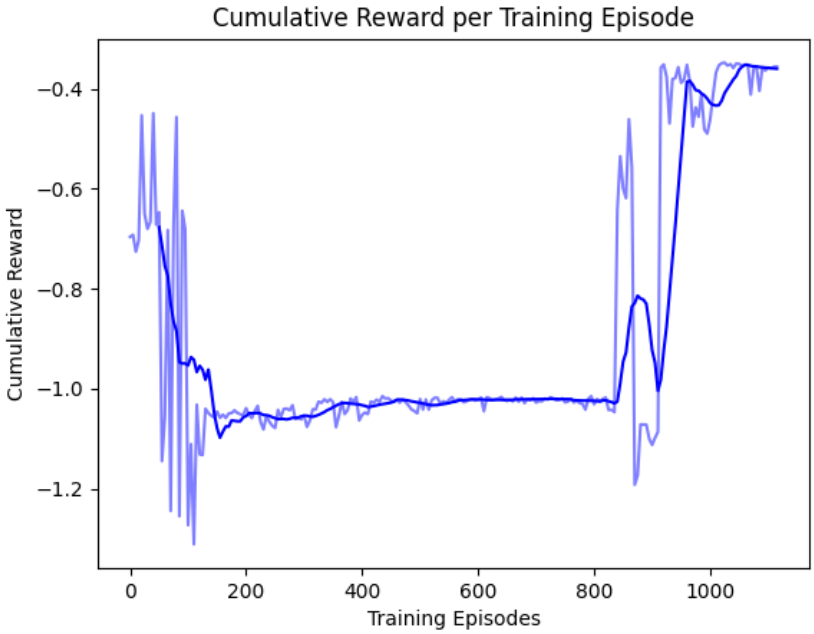


Figure 7.7: Sum of rewards per training episode. The moving average of the previous 50 episodes is shown as the dark blue line.

leads to a flow rate which is above the target flow rate, as it can be seen in Figure 7.11. Since the ball valve is required to be open at the end of an episode, the guide vanes are not allowed to be fully opened towards the end, if the target flow rate should be achieved.

Finally, we inspect how often the ball valve and air valve actions are switched and how often the guide vane opening action is -1 or 1, i.e., how often the guide vane state is changed. These quantities are given in Figure 7.12. One can see that the actions are only switched at the beginning of the simulation. After roughly 20 seconds, the actions are kept constant. Especially, the ball valve action is just switched once.

Summing it up, the investigated policy is able to achieve the objectives regarding the target rotation speed and the ball valve being open at the end. Furthermore, the number of action switches is relatively low, which leads to less wear and tear. The objective which is not met, is the target flow rate. This mainly results from fully opened guide vanes throughout most of the episode. The reason for that might be the penalty on guide vane action switches, so we also tried to exclude this objective from the reward function. Nevertheless, we were not able to reliably find a policy that achieves the target flow rate.

We believe that in order to find a policy that meets all the objectives, the structure of the reward function will need to be changed further.

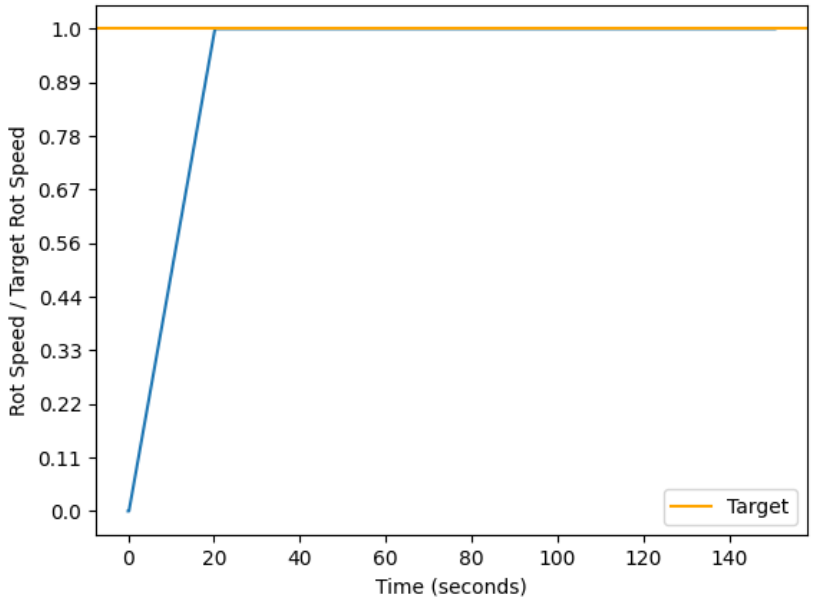
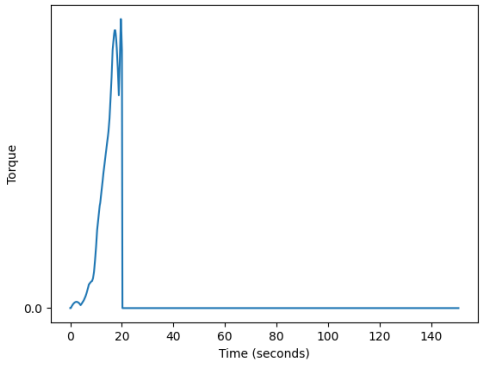
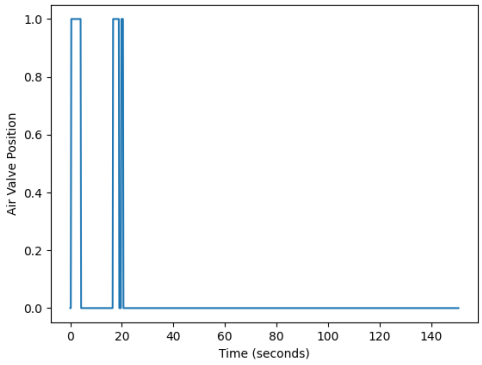


Figure 7.8: Target rotation speed is achieved fast.

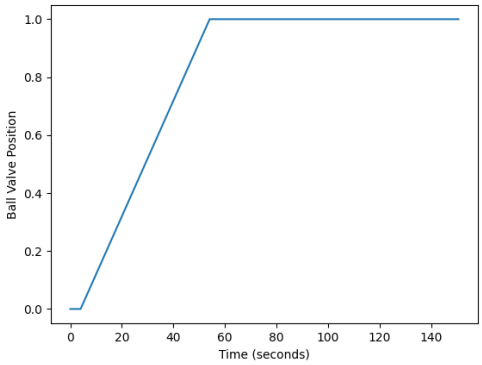


(a) Torque increases fast until the target rotation speed is met. Then the torque is constantly set to zero by the environment.

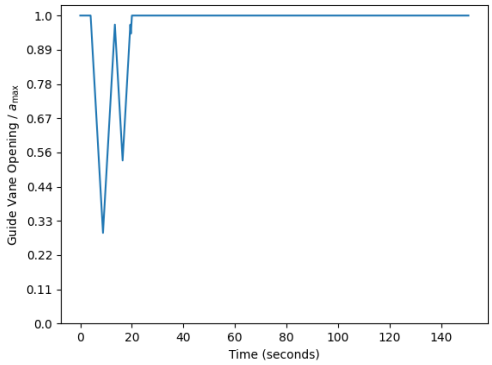


(b) The air valve is sparsely opened at the beginning, which means that only a small amount of water is displaced by air. Shortly after the target rotation speed is met, the air valve is not opened anymore. This is good since blowing in more air would have no effect, but additional costs.

Figure 7.9: Torque and air valve position for a policy that does not meet the target flow rate.



(a) The ball balve starts opening after about 5 seconds. At about 53 seconds it is completely open and stays open until the end of the episode.



(b) Guide vane opening values are varied during the first 27 seconds of the simulation. Afterwards, the guide vanes are completely open until the end of the episode.

Figure 7.10: Positions of the ball valve and guide vanes responsible for flow rate control.

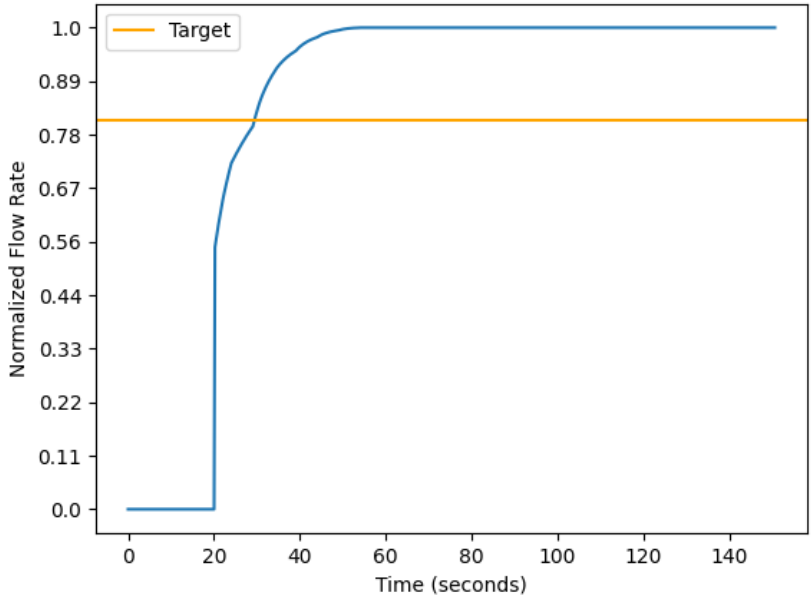
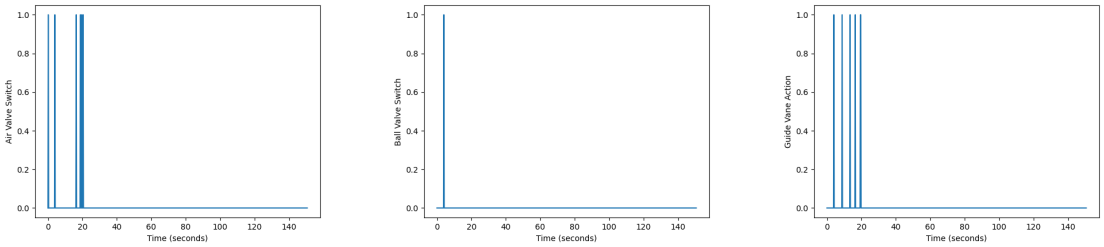


Figure 7.11: Normalized flow rate. The horizontal orange line marks the normalized target flow rate.



(a) Switches of air valve action: 0 corresponds to the action remaining the same, 1 corresponds to the action being switched.

(b) Switches of ball valve action: 0 corresponds to the action remaining the same, 1 corresponds to the action being switched.

(c) Guide vane action: 0 corresponds to action $= 0$ and 1 corresponds to action $\in \{-1, 1\}$, i.e., guide vane is further opened/closed.

Figure 7.12: Air valve, ball valve and guide vane actions: Switching actions frequently causes higher wear and tear.

8 Conclusion

We discuss several distributional RL algorithms and provide in detail pseudo-code for them. This type of algorithms provides more options to evaluate the reliability of the learnt policies. We present some methods for using the learned distributions to improve reliability. Furthermore, we discuss two approaches of off-policy evaluation which provide lower bounds for the expected return and convergence guarantees, respectively.

Then we introduce two simulation models implemented in the simulation software Simulink, which represent the environments for the RL agent. They aim to accurately model the behaviour of pump turbines which are used for pumped storage systems. Therefore, it is important to operate the pumps as reliable as possible. We find that running the simulations in Simulink is computationally expensive and makes it hard to parallelize learning. Furthermore, a lot of adjustments were required to perform RL, since the agent tries out policies and explores states, which are not visited during manual control of the pump.

While we were able to find an optimal policy for one environment, only two out of three objectives were met for the second environment.

For the first environments, the weights in the reward function had a huge impact on the training results and we believe that further adaptation of the reward function for the second environment will also lead to better learning results. This is left for future work.

In addition, it might be interesting to investigate how learning could be parallelized by implementing the simulation models in Python, or by simplifying the models while retaining their ability to represent a pump turbine in the real world.

References

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

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458, 2017. URL https://arxiv.org/abs/1707.06887.

Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2017. URL https://arxiv.org/abs/1710.10044.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning*, pages 1096–1105. PMLR, 2018. URL https://arxiv.org/abs/1806.06923.

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*, pages 6190–6199, 2019.

R. Bellman. *Dynamic Programming*. RAND Corporation research study. Princeton University Press, 1957. URL https://books.google.at/books?id=rZW4ugAACAAJ.

Thibaut Théate, Antoine Wehenkel, Adrien Bolland, Gilles Louppe, and Damien Ernst. Distributional reinforcement learning with unconstrained monotonic neural networks. *Neurocomputing*, 534:199–219, May 2023. ISSN 0925-2312. DOI: 10.1016/j.neucom.2023.02.049. URL http://dx.doi.org/10.1016/j.neucom.2023.02.049.

Cédric Villani. *Optimal Transport – Old and New*, volume 338, page 105. Springer, 2008. DOI: 10.1007/978-3-540-71050-9.

Clare Lyle, Pablo Samuel Castro, and Marc G. Bellemare. A comparative analysis of expected and distributional reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4504–4511, 2019. URL https://arxiv.org/abs/1901.11084.

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*, pages 29–37. PMLR, 2018. URL https://arxiv.org/abs/1802.08163.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. CleanRL: High-quality single-file implementations

of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23 (274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. ISSN 1476-4687. DOI: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Mastane Achab, Reda Alami, Yasser Abdelaziz Dahou Djilali, Kirill Fedyanin, and Eric Moulines. One-step distributional reinforcement learning. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://arxiv.org/abs/2304.14421.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988. ISSN 1573-0565. DOI: 10.1007/BF00115009. URL https://doi.org/10.1007/BF00115009.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Yang Peng, Liangyu Zhang, and Zhihua Zhang. Statistical efficiency of distributional temporal difference learning. In *Advances in Neural Information Processing Systems*, volume 37, pages 24724–24761. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/2c15b0221da28bc6f4373a7e78b896dd-Paper-Conference.pdf.

Mark Rowland, Rémi Munos, Mohammad Gheshlaghi Azar, Yunhao Tang, Georg Ostrovski, Anna Harutyunyan, Karl Tuyls, Marc G. Bellemare, and Will Dabney. An analysis of quantile temporal-difference learning. *Journal of Machine Learning Research*, 25(163): 1–47, 2024. URL https://arxiv.org/abs/2301.04462.

Tengyang Xie, Yifei Ma, and Yu-Xiang Wang. Towards optimal off-policy evaluation for reinforcement learning with marginalized importance sampling. *Advances in Neural Information Processing Systems*, 32, 2019. URL https://arxiv.org/abs/1906.03393.

Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. *Advances in Neural Information Processing Systems*, 31, 2018. URL https://arxiv.org/abs/1810.12429.

Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High-confidence off-policy evaluation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29 (1), Feb. 2015. DOI: 10.1609/aaai.v29i1.9541. URL https://ojs.aaai.org/index.php/AAAI/article/view/9541.

Andreas Maurer and Massimiliano Pontil. Empirical Bernstein bounds and sample variance penalization. In *Annual Conference Computational Learning Theory*, 2009. URL https://api.semanticscholar.org/CorpusID:17090214.

Runzhe Wu, Masatoshi Uehara, and Wen Sun. Distributional offline policy evaluation with predictive error guarantees. In *International Conference on Machine Learning*, pages 37685–37712. PMLR, 2023. URL https://arxiv.org/abs/2302.09456.

Kun-Jen Chung and Matthew J. Sobel. Discounted MDP's: Distribution functions and exponential utility maximization. *SIAM Journal on Control and Optimization*, 25(1): 49–62, 1987. DOI: 10.1137/0325004. URL https://doi.org/10.1137/0325004.

Guanying Chen and Zhenming Ji. A review of solar and wind energy resource projection based on the earth system model. *Sustainability*, 16(f8), 2024. ISSN 2071-1050. DOI: 10.3390/su16083339. URL https://www.mdpi.com/2071-1050/16/8/3339.

Simulink Documentation. Simulation and model-based design, 2020. URL https://www.mathworks.com/products/simulink.html.

Carlotta Tubeuf, Felix Birkelbach, Anton Maly, and René Hofmann. Increasing the flexibility of hydropower with reinforcement learning on a digital twin platform. *Energies*, 16(4), 2023. ISSN 1996-1073. DOI: 10.3390/en16041796. URL https://www.mdpi.com/1996-1073/16/4/1796.

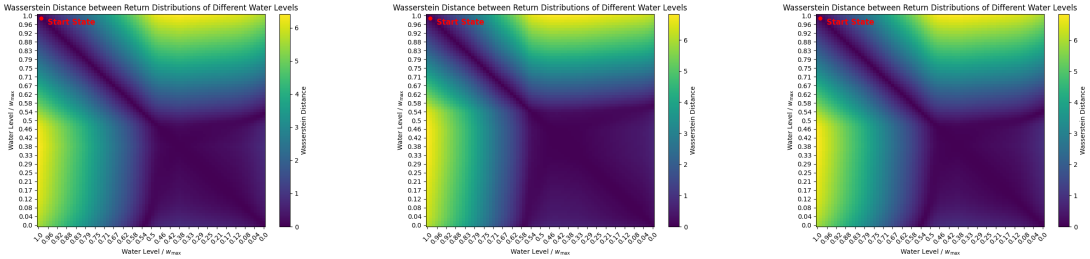
Carlotta Tubeuf, Jakob aus der Schmitten, René Hofmann, Clemens Heitzinger, and Felix Birkelbach. Improving control of energy systems with reinforcement learning: Application to a reversible pump turbine. In *Energy Sustainability*, volume 87899, page V001T01A001. American Society of Mechanical Engineers, 2024. DOI: 10.1115/ES2024-122475. URL https://doi.org/10.1115/ES2024-122475.

Johannes Brust. Simulink Gym, 2025. URL https://github.com/johbrust/simulink_gym.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013. ISSN 1076-9757. DOI: 10.1613/jair.3912. URL http://dx.doi.org/10.1613/jair.3912.

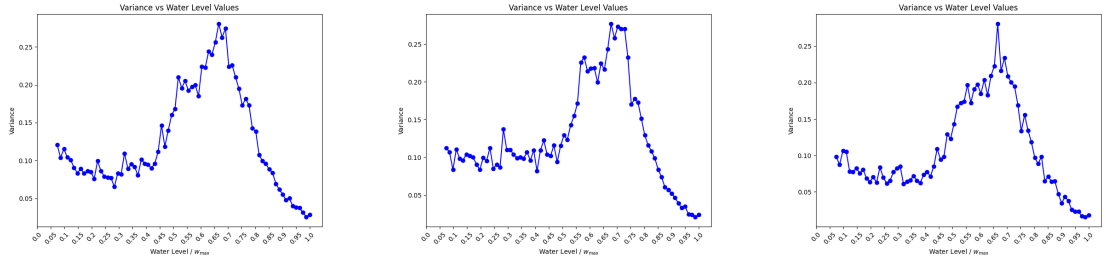
Appendix

In the following, the supplementary plots from Section 7.2 are shown.



(a) $s_{\text{prev action}} = 0$, action $a = 0$ (b) $s_{\text{prev action}} = 1$, action $a = 0$ (c) $s_{\text{prev action}} = 0$, action $a = 1$

Figure 8.1: Pairwise comparison of return distributions for varying initial state-action pairs. The difference is measured in Wasserstein distance.



(a) $s_{\text{prev action}} = 0$, action $a = 0$ (b) $s_{\text{prev action}} = 1$, action $a = 0$ (c) $s_{\text{prev action}} = 0$, action $a = 1$

Figure 8.2: Variances of return distributions for varying initial state-action pairs.