QUANTILE REGRESSION REINFORCEMENT LEARNING WITH STATE ALIGNED VECTOR REWARDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning from a scalar reward in continuous action space environments is difficult and often requires millions if not billions of interactions. We introduce state aligned vector rewards, which are easily defined in metric state spaces and allow our deep reinforcement learning agent to tackle the curse of dimensionality. Our agent learns to map from action distributions to state change distributions implicitly defined in a quantile function neural network. We further introduce a new reinforcement learning technique inspired by quantile regression which does not limit agents to explicitly parameterized action distributions. Our results in high dimensional state spaces show that training with vector rewards allows our agent to learn multiple times faster than an agent training with scalar rewards.

1 INTRODUCTION

Reinforcement learning (Sutton et al., 1998) is a powerful paradigm in which an agent learns about an environment through interaction. The common formulation consists of a Markov Decision Process (MDP) modeled as a 5-tuple $(\mathbb{S}, \mathbb{A}, P, r, \gamma)$ where $\mathbb{S}$ is the (possibly infinite) set of states, $\mathbb{A}$ is the (possibly infinite) set of actions available to the agent, $P : (\mathbb{S} \times \mathbb{A} \times \mathbb{S}) \rightarrow [0, 1] : P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ is the transition probability of reaching state $\boldsymbol{s}' \in \mathbb{S}$ given state $\boldsymbol{s} \in \mathbb{S}$ and action $\boldsymbol{a} \in \mathbb{A}$, $r : (\mathbb{S} \times \mathbb{A}) \rightarrow \mathbb{R} : r(\boldsymbol{s}, \boldsymbol{a})$ is the reward received for taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$ and γ is the reward discount factor. The goal of the agent is to maximize the cumulative discounted reward $R = \sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ by choosing actions $\boldsymbol{a}_t$ according to some (possibly stochastic) policy $\pi : (\mathbb{S} \times \mathbb{A}) \rightarrow [0, 1] : \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. Sometimes it is further useful to make a distinction between the actual state space $\mathbb{S}$ and the correlated observation space $\mathbb{O}$ of the agent. In this case $\pi : (\mathbb{O} \times \mathbb{A}) \rightarrow [0, 1] : \pi(\boldsymbol{a}_t|\boldsymbol{o}_t)$ with $\boldsymbol{o}_t \in \mathbb{O}$. The use of deep neural networks allowed this formulation to scale to high dimensional visual inputs approaching continuity in state space (Mnih et al., 2015) while others extended deep reinforcement learning to continuous action spaces (Lillicrap et al., 2015; Mnih et al., 2016). While neural networks are powerful function approximators, they require large amounts of training data to converge. In the case of reinforcement learning this means interactions with the environment, a requirement easy to fulfill in simulation, yet impractical when the agent should interact with the real world. This problem is aggravated by the weak training signal of classical reinforcement learning – a simple scalar reward.

While originally the dopamine activity in mammal brains was linked to general "rewarding" events, Pinto & Lammel (2017) point out that the diversity of dopamine circuits in the mid brain is better modeled by viral vector strategies. Gershman et al. (2009) also show that human reinforcement learning incorporates effector specific value estimations to cope with the high dimensional action space. Inspired by these biological insights, we improve the sample efficiency of a deep reinforcement learning algorithm in this work by modeling a d-dimensional vector reward. A vector reward can in some domains easily be defined in alignment with the state space. We say that two vector spaces are *aligned* if their dimensions correlate and show that if state and action space are *not* aligned, a mapping from action distribution to state change distribution can be learned.

As a motivating example, consider the agent in Figure 1(a) trying to reach the goal (marked by the blue dot). If we take $\boldsymbol{p}$ as the position vector of the agent relative to the goal, a sensible reward to guide the agent to the goal in this environment would be $r = ||\boldsymbol{p}|| - ||\boldsymbol{p} + \boldsymbol{a}||$ where $||\cdot||$ can be any norm in the vector space of the environment. For illustration purposes we'll focus on the L^1 norm in this example and throughout this paper. During training the agent might try action $\boldsymbol{a}$ which moves

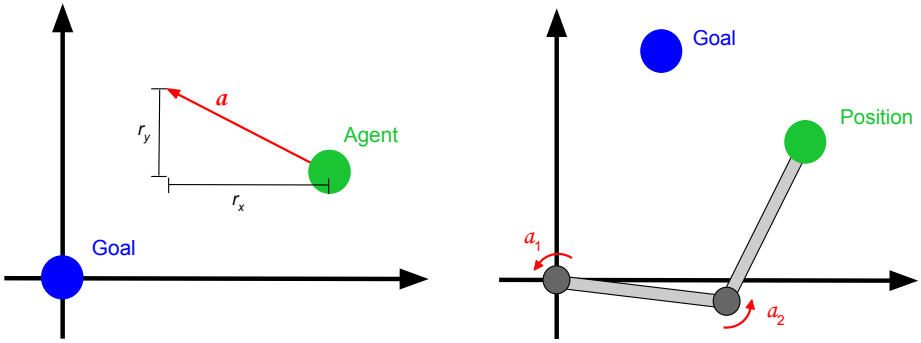

Figure 1: (a) An agent freely moving in a 2D world might try to reach a goal at position (0, 0) by taking action $\boldsymbol{a}$. A sensible reward in this environment is the change in absolute distance to the goal. With a scalar reward this would be summarized as $r = r_x - r_y$, whereas a vector reward would keep the two reward dimensions distinguishable. (b) In most cases action and state space are however not aligned, therefore a mapping from action to state change must be learned.

it closer to the goal in x direction, but a bit further away from the goal in y direction. The scalar reward would then just convey the information, that the action was rather positive (since the agent got closer to the goal) but miss out on the distinction that the action was good in x-direction but bad in y-direction. To provide this distinction, a more informative reward would keep the dimensions separate and therefore be a vector itself: $\boldsymbol{r} = |\boldsymbol{p}| - |\boldsymbol{p} + \boldsymbol{a}|$ where $|\cdot|$ denotes the element-wise absolute value here. Note that this reward is dimension wise aligned with the position $\boldsymbol{p}$, the *state*, of the agent. Since we focus on reaching problems in this work, we'll use the terms "position" and "state" interchangeably.

The problem with such a state aligned vector reward is however that the action space is in most cases not state aligned. To see this, consider the schematic robot arm in Figure 1(b): The action dimensions a_1 and a_2 correspond to the torques of the robot arm and do not directly translate to a shift in x and y dimension, respectively. To address this issue we use the method proposed by Dabney et al. (2018b;a) to train a deep neural network to approximate the quantile function, in our case of the position change, given the current observation and quantile target. Additionally we give a parameterization of the action probability distribution as input to this position change prediction network (short PCPN). We then train the agent, parameterized by another neural network which maps from observations to action probability distributions, through a new reinforcement learning method we call quantile regression reinforcement learning (short QRRL). A schematic overview of our setup can be seen in Figure 2.

To summarize, the contributions of this paper are the following:

- We extend the reinforcement learning paradigm to allow for faster training based on more informative state aligned vector rewards.
- We present an architecture that learns a probability distribution over possible state changes based on a probability distribution over possible actions.
- We introduce a new reinforcement learning algorithm to train stochastic continuous action policies with arbitrary action probability distributions.

2 Quantile Regression and Implicit Quantile Networks

Quantile regression (Koenker, 2005) discusses approximation techniques for the inverse cumulative distribution function F_Y^{-1}, i.e., the *quantile function*, of some probability distribution Y. Recent work (Dabney et al., 2018a; Ostrovski et al., 2018) shows that a neural network can learn to approximate the quantile function by mapping a uniformly sampled quantile target $\tau \sim \mathcal{U}([0, 1])$ to its corresponding quantile function value $F_Y^{-1}(\tau) \in \mathbb{R}$. Thereby the trained neural network implicitly models the full probability distribution Y.

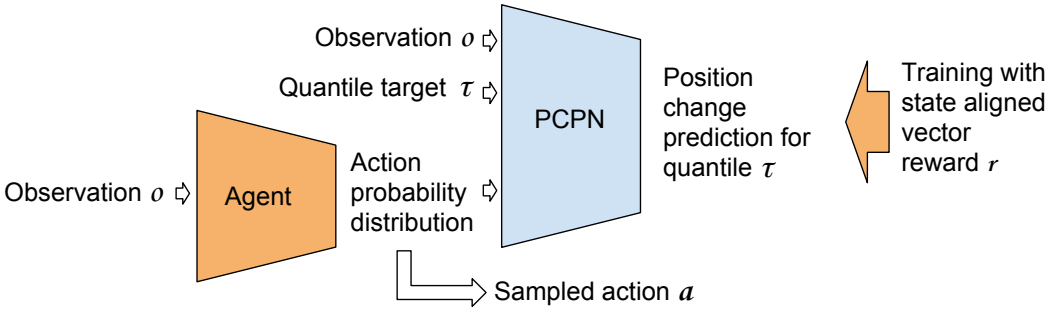

Figure 2: An overview of the architecture setup. The agent and the position change prediction network (PCPN) are instantiated with neural networks and trained through quantile regression reinforcement learning and quantile regression, respectively.

More formally, let $W_p(U, Y)$ be the p-Wasserstein metric $W_p(U, Y) = \left(\int_0^1 |F_Y^{-1}(\omega) - F_U^{-1}(\omega)|^p d\omega\right)^{1/p}$ of distributions U and Y, also characterized as the L^p metric of quantile functions (Müller, 1997). Dabney et al. (2018b) show that the quantile regression loss (Koenker & Hallock, 2001)

$$\rho_\tau(\delta) = (\tau - \mathbf{1}_{\delta<0}) \cdot \delta \tag{1}$$

minimizes the 1-Wasserstein distance of a scalar probability distribution Y to a uniform mixture of Diracs U. Here, $\delta = y - u$ with $y \sim Y$ and $u \sim U$ is the quantile sample error. Ostrovski et al. (2018) generalized this result by showing that the expected quantile loss $\mathbb{E}_{\tau\sim\mathcal{U}([0,1])}\left[\mathbb{E}_{z\sim Z}\left[\rho_\tau(z - \hat{Q}_\theta(\tau))\right]\right]$ of a parameterized quantile function $\hat{Q}_\theta$ aproximating the quantile function F_Z^{-1} of some distribution Z is equal to the quantile divergence

$$q(Z, Q_\theta) := \int_0^1 \left[\int_{F_Z^{-1}(\tau)}^{F_{Q_\theta}^{-1}(\tau)} (F_Z(x) - \tau) dx\right] d\tau$$

plus some constant not depending on the parameters θ. Here, Q_θ is the distribution implicitly defined by $\hat{Q}_\theta$. Therefore, training a neural network $\hat{Q}_\theta(\tau)$ to minimize $\rho_\tau(z - \hat{Q}_\theta(\tau))$ with z sampled from the target probability distribution Z effectively minimizes the quantile divergence $q(Z, Q_\theta)$ and thereby models an approximate distribution Q_θ of Z implicitly in the network parameters θ of the neural network $\hat{Q}_\theta(\tau)$.

By approximating the quantile function instead of a parameterized probability distribution, as common in many deep learning models (Kingma & Welling, 2013; Lillicrap et al., 2015; Ha & Schmidhuber, 2018), we do not enforce any constraint on the probability distribution Z, e.g., Z can be multi-modal, not continuous and non-Gaussian. This is crucial for our case as a position change distribution given a certain action distribution can have any shape.

In our setup, we train the PCPN with the quantile regression loss (1) to approximate a position change quantile function per position dimension. Aside from a target quantile $\tau \sim \mathcal{U}([0, 1])$ per position dimension, the network input consists of an observation $\boldsymbol{o} \in \mathbb{O}$ and a multi-variate Gaussian action probability distribution $A = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with diagonal covariance matrix $\boldsymbol{\Sigma} = \boldsymbol{I}\boldsymbol{\sigma}$, parameterized by the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}$ vectors. Therefore, the PCPN has the ability to implicitly learn the conditional position change distribution given an observation and an action distribution.

3 Quantile Regression Reinforcement Learning

The dominant reinforcement learning algorithms are either value based methods, e.g., Q-learning (Watkins, 1989; Mnih et al., 2015), policy gradient based methods, e.g., REINFORCE (Williams, 1992; Sutton et al., 1999), or a combination of both, e.g., actor-critic methods (Sutton et al., 1998; Mnih et al., 2016). In this section we establish a new reinforcement learning objective based on

quantile regression. In contrast to Q-learning we allow for a continuous action space similar to policy based algorithms, but in contrast to policy based algorithms we do not limit ourselves to explicitly parameterized policies. Note that the technique described in this section is generally applicable to reinforcement learning problems and we therefore use the terms "action" and "policy" in the common reinforcement learning meaning, which is distinct from the meaning of "action" and "policy" in the rest of the paper. We later link in Section 4 the position change estimation of the PCPN to this new meaning of "action" to connect with the rest of the paper.

The main idea behind Quantile Regression Reinforcement Learning (or QRRL in short) is to model the policy implicitly in the network parameters, therefore allowing for complex stochastic policies which can find multi-modal stochastic optima in policy space. For this we model for each action dimension the quantile function $\hat{Q}_\theta(\tau, \boldsymbol{o}) = F^{-1}_{Q_\theta}(\tau, \boldsymbol{o})$ of the implicitly defined action distribution $Q_\theta(\boldsymbol{o})$ by a neural network with an observation $\boldsymbol{o}$ and a target quantile $\tau \in [0, 1]$ as input. From the full network $\hat{\boldsymbol{Q}}_\xi : [0, 1]^d \times \mathbb{O} \to \mathbb{R}^d$, with d being the number of action dimensions, an action $\boldsymbol{a} \in \mathbb{A} \equiv \mathbb{R}^d$ for a given observation $\boldsymbol{o}$ can be sampled by sampling $\boldsymbol{\tau} \sim \mathcal{U}([0, 1]^d)$ and taking the network output as action. Since the network approximates quantile functions, the network output of a uniformly at random sampled quantile target is a sample from the implicitly defined action distribution. The question left to address is how to train the network, such that it (a) approximates the quantile functions of the action dimensions and (b) the implicitly defined policy maximizes the expected (discounted) reward R.

Here quantile regression comes in handy. Informally put, quantile regression is linked to the Wasserstein metric which is also sometimes refered to as *earth movers distance*. Imagine a pile of earth representing probability mass. In reinforcement learning we essentially want to move probability mass towards actions that were good and away from actions that were bad, where "good" and "bad" are measured by discounted accumulative (bootstrapped) reward achieved. Quantile regression can achieve this neatly by shaping the pile of earth according to an advantage estimation and the constraint of monotonicity (which is a core property of quantile functions).

More formally, we are interested in the effect of the quantile regression loss (1) on action probabilities when the error δ is between two samples from the same network, one representing the action that was played in the environment (which resulted from some sample quantile τ) and one representing the quantile function value at some τ' where the network tries to approximate the quantile function of the optimal action distribution. We focus in this analysis on the simple case of a single action dimension with scalar quantile target input and scalar quantile function output. The generalization to multidimensional action-/quantile-functions with independent action dimensions follows trivially.

First, let us denote the action taken in the environment by $a_\tau := \hat{Q}_\theta(\tau, \boldsymbol{o}) = \hat{Q}_\theta(\tau)$, where we hide the dependence of $\hat{Q}_\theta(\tau, \boldsymbol{o})$ on the observation $\boldsymbol{o}$ hereafter for notation simplicity. To train the quantile network we sample different τ' and consider the loss

$$\rho_{\tau'}(a_\tau - \hat{Q}_\theta(\tau')) = (\tau' - \mathbf{1}_{a_\tau - \hat{Q}_\theta(\tau') < 0}) \cdot (\tau - \hat{Q}_\theta(\tau')).$$

As we are interested in the effect of the loss on the probability of a_τ consider a τ' close to τ, i.e., $\tau' = \tau \pm \epsilon$ for some $\epsilon > 0$ and $\epsilon < \tau < 1 - \epsilon$. For $\tau' = \tau - \epsilon$ the loss reduces to

$$\rho_{\tau'}(a_\tau - \hat{Q}_\theta(\tau - \epsilon)) = (\tau - \epsilon) \cdot (a_\tau - \hat{Q}_\theta(\tau - \epsilon)) = \epsilon(\tau - \epsilon) \cdot \frac{a_\tau - \hat{Q}_\theta(\tau - \epsilon)}{\epsilon} \xrightarrow{\epsilon \to 0} d\tau_1 \frac{\delta \hat{Q}_\theta(\tau)}{\delta \tau}$$

where $d\tau_1$ is an infinitely small but positive value Similarly we get for $\tau' = \tau + \epsilon$

$$\rho_{\tau'}(a_\tau - \hat{Q}_\theta(\tau + \epsilon)) = (\tau + \epsilon - 1) \cdot (a_\tau - \hat{Q}_\theta(\tau + \epsilon)) = \epsilon(1 - \tau - \epsilon) \cdot \frac{\hat{Q}_\theta(\tau + \epsilon) - a_\tau}{\epsilon} \xrightarrow{\epsilon \to 0} d\tau_2 \frac{\delta \hat{Q}_\theta(\tau)}{\delta \tau}$$

with $d\tau_2$ an infinitely small but positive value. Therefore, the quantile regression loss is positively correlated to the slope of $\hat{Q}_\theta$ at τ. The partial derivative of a quantile function $\hat{Q}_\theta(\tau)$ with respect to the quantile τ is however also known as the *sparsity function* (Tukey, 1965) or *quantile-density function* (Parzen, 1979) and has the interesting property (Jones, 1992):

$$\frac{\delta}{\delta \tau} \hat{Q}_\theta(\tau) = \frac{1}{p_{Q_\theta}(\hat{Q}_\theta(\tau))} = \frac{1}{p_{Q_\theta}(a_\tau)}$$

where p_{Q_θ} is the probability density function of distribution Q_θ. Hence, the quantile regression loss is inverse proportional to the probability of action a_τ, which implies the following, given that $\hat{Q}_\theta$ is the quantile function of distribution Q_θ:

1. Minimizing the quantile loss (1) for a given action a_τ increases the likelihood of action a_τ.
2. Maximizing the quantile loss for a given action a_τ decreases the likelihood of action a_τ.

This leads us to the QRRL actor objective

$$\min_\theta \mathbb{E}_{a_\tau}[\mathbb{E}_{\tau'}[\rho_{\tau'}(a_\tau - \hat{Q}_\theta(\tau', \boldsymbol{o}_t)) \cdot (R_{t,n} - V_\psi(\boldsymbol{o}_t))]] \text{ s.t. } \hat{Q}_\theta \text{ is monotonically increasing with } \tau'$$

where $(R_{t,n} - V_\psi(\boldsymbol{o}_t))$ is an advantage estimation (Mnih et al., 2016) with $R_{t,n} = \gamma^n V_\psi(\boldsymbol{o}_{t+n}) + \sum_{t'=t}^{t+n} \gamma^{t'-t} r_{t'}$ being the n-step estimate of the discounted future reward and $V_\psi(\cdot)$ being the output of a critic network with parameters ψ, which is trained to approximate $R_{t,n}$. Note that QRRL is a constraint optimization (since $\hat{Q}_\theta$ must be a proper quantile function, i.e., monotonically increasing). In our experiments we found it however sufficient to add this constraint as an additional loss term

$$\mathcal{L}_{mon}(\tau, \tau', \theta) = \mathcal{L}^\kappa_{Huber}(\hat{Q}_\theta(\tau) - \hat{Q}_\theta(\tau')) \cdot \mathbf{1}_{\tau<\tau' \,\&\, \hat{Q}_\theta(\tau)>\hat{Q}_\theta(\tau') \text{ or } \tau>\tau' \,\&\, \hat{Q}_\theta(\tau)<\hat{Q}_\theta(\tau')}$$

$$\text{with } \mathcal{L}^\kappa_{Huber}(\delta) = \begin{cases} \delta^2 & |\delta| \leq \kappa \\ |\delta| - \frac{1}{2}\kappa & \text{otherwise} \end{cases}$$

We weight this additional loss term with a constant Lagrange multiplier λ_{mon} in the full loss term

$$\mathcal{L}_{QRRL}(a_\tau, \boldsymbol{o}_t, \tau, \tau', \theta, \psi) = \mathcal{L}_a(a_\tau, \boldsymbol{o}_t, \tau', \theta) + \lambda_c \mathcal{L}_c(\boldsymbol{o}_t, \psi) + \lambda_{mon} \mathcal{L}_{mon}(\tau, \tau', \theta) \tag{2}$$

with $\mathcal{L}_a(a_\tau, \boldsymbol{o}_t, \tau', \theta) = \rho_{\tau'}(a_\tau - \hat{Q}_\theta(\tau', \boldsymbol{o})) \cdot (R_{t,n} - V_\psi(\boldsymbol{o}_t))$ and $\mathcal{L}_c(\boldsymbol{o}_t, \psi) = (R_{t,n} - V_\psi(\boldsymbol{o}_t))^2$. λ_c is another constant Lagrange multiplier to weight the critic loss $\mathcal{L}_c$ against the actor loss $\mathcal{L}_a$. Although we focus in our experiments on an on-policy method with multiple actors, QRRL can easily be adapted to the off-policy setting.

4 State Aligned Vector Reward Agent

In our State Aligned VEctor Reward (SAVER) agent, we use QRRL to train the agent network AN_η *through* the PCPN. For this we feed the action probability distribution output $\boldsymbol{\mu}, \boldsymbol{\sigma}$ of the agent network to a pretrained PCPN and train on the QRRL loss (2) with respect to the agent network parameters η, where we take the actual position change $\boldsymbol{\Delta p}$ introduced by a sampled action $\boldsymbol{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I\sigma})$ as QRRL action target $\boldsymbol{a}_\tau = \boldsymbol{\Delta p}$ and compare it to K sampled position change estimations $\boldsymbol{\Delta \hat{p}}(\boldsymbol{\tau}^{(i)}), \boldsymbol{\tau}^{(i)} \sim \mathcal{U}([0,1]^d)$ for $i \in \{1, ..., K\}$. Note that we pretrain the PCPN in our setup with random observations and action probability distributions and freeze the PCPN weights during agent training. Therefore, one could potentially train several agents to solve different tasks in the same environment using the same PCPN.

Since the output of the PCPN can be seen as state aligned action, training on vector rewards is straight forward. Instead of having a scalar critic estimating the value function $V(\cdot) \in \mathbb{R}$ we estimate a value per action dimension, i.e., $\boldsymbol{V}(\cdot) \in \mathbb{R}^d$. Similarly, the vector rewards can be summed to a vector n-step discounted reward estimation $\boldsymbol{R}_{t,n} = \gamma^n \boldsymbol{V}(\boldsymbol{o}_{t+n}) + \sum_{t'=t}^{t+n} \gamma^{t'-t} \boldsymbol{r}_{t'}$, which leads us to a vector advantage $(\boldsymbol{R}_{t,n} - \boldsymbol{V}(\boldsymbol{o}_t))$ at timestep t. Therefore we have an advantage estimation for each position change dimension individually and can apply loss (2) to each position change estimation dimension individually.

5 Experiments and Results

To test our ideas, we implement three simple experiments, two to test our approach in high dimensional metric spaces and one to show the applicability of our approach to a real world problem by modeling the robot arm shown in Figure 1(b). For reproducibility and to stimulate further research in this area, our code is publicly available.[1] We compare our SAVER agent against an A2C agent, the synchronous variant of A3C (Mnih et al., 2016). We choose this baseline since it is most similar to our SAVER implementation and we believe that SAVER can benefit in the future from state-of-the-art additions to A3C as presented by Espeholt et al. (2018). Here however, we want to focus on the benefit of using vector rewards instead of scalar rewards. We implement a simple feed forward

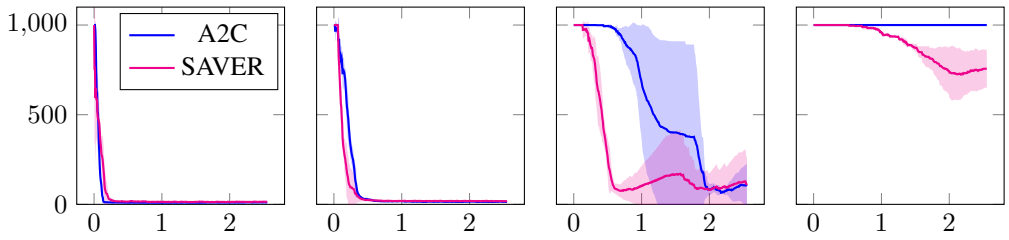

Figure 3: Average number of steps needed to reach the goal over the course of training of an agent that chooses pairwise angle and step size for pairwise grouped dimensions. Results are shown for a 2, 4, 8 and 16 dimensional environment, corresponding plots are ordered from left to right. The x-axis shows the number of training steps in millions while the y-axis shows the average episode length over the last 100 episodes. Training length was fixed to 2,560,000 steps. Plotted is the average of 3 training runs with the shaded area indicating the standard deviation between runs. The plots suggest that the higher dimensional the environment, the more apparent the gain of training on vector rewards is.

neural architecture architecture and fix most hyperparameters to values that work well for SAVER and A2C. Details can be found in Appendix A.

In a first experiment, we let the agent move freely in all directions within a d-dimensional hypercube by choosing the action space to represent a set of (angle, step size) pairs for moving in the pairwise grouped environment dimensions. Note that this setup is similar to the depiction in Figure 1(a), but the action dimensions are not aligned with the state-/position-change. The initial position of the agent is sampled uniformly at random from $\mathcal{U}([-1,1]^d)$ and the goal of the agent is to reach the origin of the d-dimensional space. The agent's observation is its position and movement of the agent is stopped at the boundaries of the hypercube defined by $[-1,1]^d$. Step sizes are re-scaled and clipped to a maximal value of 0.1 and episodes are terminated after 1,000 steps if the agent didn't manage to get into close proximity of the goal beforehand. We pretrain the PCPN with 100,000 batches of 128 transitions each by randomly sampling $\boldsymbol{o} \sim \mathcal{U}([-1,1]^d)$, $\boldsymbol{\mu} \sim \mathcal{U}([-1,1]^d)$ and $\boldsymbol{\sigma} \sim \mathcal{U}([0,1]^d)$ where actions between -1 and 1 correspond to step sizes between -0.1 and 0.1 and angles between 0 and π, respectively.[2] We performed a small hyperparameter search for the learning rate in the 8-dimensional hypercube and settled for a learning rate of 0.01 (chosen from $\{0.01, 0.003, 0.001\}$) for SAVER and a learning rate of 0.0003 (chosen from $\{0.001, 0.0003, 0.0001\}$) for A2C.[3] We measure the performance of the agents by the mean length of the last 100 episodes. This mean episode length is plotted in Figure 3 over the course of training for hypercubes of different dimensionality. Average and standard deviation of three training runs is shown. As can be seen from the plots, the higher the dimensionality d of the environment, the more apparent the advantage of training on vector rewards gets. SAVER trains faster in high dimensional cubes and is even able to find the goal in a 16-dimensional cube given the step limit of 1,000 steps.

As a second experiment, we keep the environment specifications the same but change the action representation. Here the agent's action consists of two parts: a softmax distribution from which the dimension to be manipulated is chosen (discrete action part) and a scalar Gaussian distribution defined by network outputs μ and σ from which the step size is sampled (continuous action part). Note that this action composition leads to a not continuous position change distribution when regarding the position change dimensions isolated, since in each dimension the probability of a change equal to 0 is more likely then position changes close to 0. We show with this experiment, that the quantile regression based PCPN can learn to implicitly model this complex position change distribution based on the composed action distribution input. For this we pretrain the PCPN with 10,000 batches of 128 transitions each by sampling softmax logits and μ uniformly at random from $\mathcal{U}([-1,1])$ and

[1] Our code can be found at: https://goo.gl/rMuadg (Anonymous Google Drive for double blind review)

[2] Note that even though SAVER requires pretraining, pretraining is task independent and other tasks could be learned by reusing a trained PCPN. Therefore we do not include the pretraining in our plots. In our experiments however, we retrain the PCPN for each training run to show algorithmic stability.

[3] We found in informal early experiments, that A2C requires a much smaller initial learning rate than the QRRL based SAVER for stable learning.

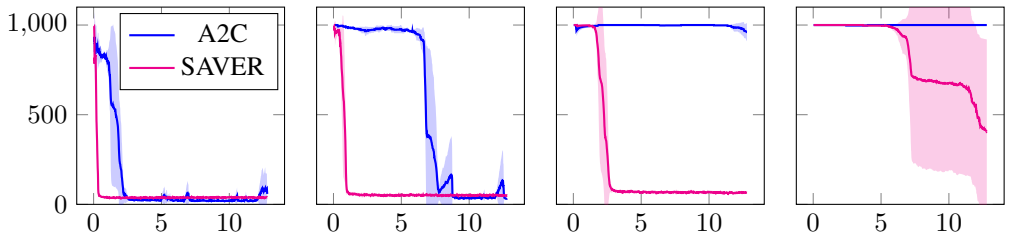

Figure 4: Average number of steps needed to reach the goal over the course of training of an agent that can select the dimension it wants to move in and the amount by how much it wants to move. Results are shown for a 3, 4, 5 and 6 dimensional environment, corresponding plots are ordered from left to right. The x-axis shows the number of training steps in millions while the y-axis shows the average episode length over the last 100 episodes. Training length was fixed to 12,80,000 steps. Plotted is the average of 3 training runs with the shaded area indicating the standard deviation between runs. Again we see a benefit of training on vector rewards in high dimensional environments.

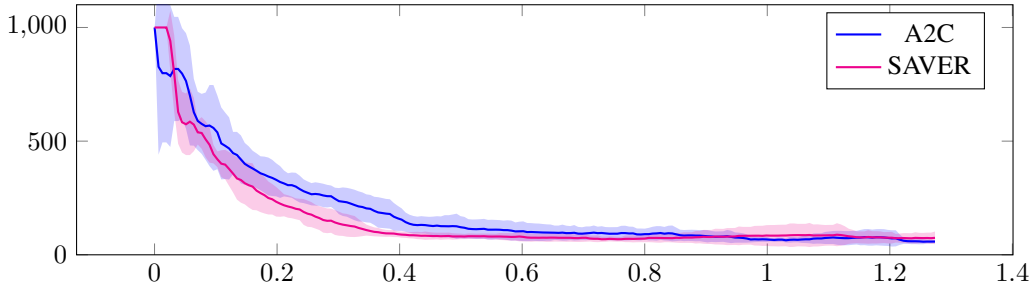

Figure 5: Average number of steps needed to reach the goal over the course of training of an agent that controls a robot arm as depicted in Figure 1(b). The x-axis shows the number of training steps in millions while the y-axis shows the average episode length over the last 100 episodes. Plotted is the average of 5 training runs with the shaded area indicating the standard deviation between runs.

σ from $\mathcal{U}([0, 1])$. In this experiment we searched for an appropriat learning rate in the 4 dimensional hypercube and settled for 0.01 for SAVER (chosen from $\{0.01, 0.003, 0.001\}$) and 0.0001 for A2C (chosen from $\{0.001, 0.0003, 0.0001\}$). The corresponding mean episode lengths over the course of agent training are plotted in Figure 4. Again we find that the higher dimensional the hypercube is, the more advantageous it is to train with vector rewards.

In our last experiment, we model the 2-joint robot arm depicted in Figure 1(b). Here an action sampled from the 2-dimensional Gaussian $\boldsymbol{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I\sigma})$ is translated into an angle change between $-0.02 \cdot \pi$ and $0.02 \cdot \pi$ of the joints. The agent's observation $\boldsymbol{o}$ consist here additionally to the hand position also of a goal position and the current angle of the joints. At the beginning of each episode, the initial joint angles and the goal position are randomly sampled from $\mathcal{U}([-\pi, \pi]^2)$ and $\mathcal{U}([-1, 1]^2)$, respectively. We choose the learning rate of SAVER as 0.003 and A2C as 0.0003 based on a search over $\{0.1, 0.01, 0.003, 0.001\}$ and $\{0.01, 0.001, 0.0003, 0.0001\}$, respectively. We pretrain the PCPN with 10,000 batches of 128 transitions each by sampling $\boldsymbol{\mu} \sim \mathcal{U}([-1, 1]^d)$ and $\boldsymbol{\sigma} \sim \mathcal{U}([0, 1]^d)$. The corresponding mean episode lengths over the course of training are plotted in Figure 5. Even in this low dimensional problem, SAVER trains slightly faster than A2C due to the more informative vector rewards.

6 Related Work

Using a vector of rewards instead of a scalar reward is most common in the literature on Multi-Objective Reinforcement Learning (MORL) (Liu et al., 2015) and Multi-Objective Sequential Decision Making (Roijers et al., 2013; Roijers & Whiteson, 2017). However, since most of this literature focuses on classical reinforcement learning and conflicting objectives, the methods discussed either

reduce the multiple objectives to a single objective or learn different policies for each objective additionally to a superpolicy deciding on which policy to use when. Recent work (Mossalam et al., 2016; Tajmajer, 2017; Nguyen, 2018) also applied these techniques to deep reinforcement learning agents. In contrast, we directly use the vector reward as more informative training signal for a neural network which implicitly learns to trade off different objectives.

Other adjacent areas are multi-agent and multi-task deep reinforcement learning (Arulkumaran et al., 2017). While former considers multiple agents with potentially different observations, later addresses how a single agent best solves multiple tasks. In contrast, our work considers a single agent solving a single task by leveraging a multi-dimensional reward.

Klinkhammer (2018) discusses multiple problem aligned rewards for better learning, while Brys et al. (2014) discuss the effect of correlated rewards on learning performance. Brys et al. (2014) also suggest multiple reward shapings of the same reward function for faster learning. Van Seijen et al. (2017) decompose the reward function into multiple rewards and train an architecture similar to ours with deep Q learning, assigning a Q-value output to each reward. While all three approaches come to the same conclusion as we do, i.e., increased training performance, they do require hand engineered reward functions, reward shapings and/or reward decompositions. In contrast, our approach is based on the fact that many state spaces are metric spaces and therefore allow for a straight forward vector reward interpretation. This makes our approach easier to apply to a larger set of tasks. While state proximity was already used by McCallum (1992) for faster backpropagation of rewards in tabular reinforcement learning, we are unaware of any deep learning algorithm capitalizing on state aligned vector rewards as we do.

Many recent approaches to deep reinforcement learning learn the environment dynamics to have a richer training signal (Dosovitskiy & Koltun, 2016), imagine action consequences (Racanière et al., 2017), improve exploration (Pathak et al., 2017) or dream up solutions (Ha & Schmidhuber, 2018). An interesting line of research (Zhang et al., 2017; Barreto et al., 2017; Ma et al., 2018; Barreto et al., 2018) in this direction uses successor features to share knowledge between tasks in the same environment. In contrast to most of these works, which often only predict one possible next state or successor feature, our PCPN incorporates the full probability distribution of possible state changes. While Ha & Schmidhuber (2018) also predict the full probability distribution of their next state representation by a Gaussian mixture model, our approach is more general in that it is also able to approximate non-Gaussian probability distributions.

Dabney et al. (2018b) were the first to use quantile regression in connection with deep reinforcement learning. In their work, including their followup work (Dabney et al., 2018a), they focused on approximating the full probability distribution of the value function. In contrast, with QRRL we explore possibilities of using quantile regression to approximate richer policies by not constraining the action distribution to an explicitly parameterized distribution. Ostrovski et al. (2018) showed that quantile networks can also be used for generative modeling. In general, we see quantile regression in combination with deep learning to have a lot of potential for future work.

7 Conclusion

In this work we present the idea of state aligned vector rewards for faster reinforcement learning. While the idea is straight forward and simple, we are unaware of any work that addresses it so far. Additionally, we also present a new reinforcement learning technique based on quantile regression in this work which we term QRRL. QRRL allows for complex stochastic policies in continuous action spaces, not limiting agents to Gaussian actions. Combining both, we show that the agent network in our SAVER agent can be trained through a quantile network pretrained in the environment. We show that SAVER is capable of training orders of magnitudes faster in high dimensional metric spaces. While d-dimensional metric spaces are mainly mathematical constructs for $d > 3$, we see a lot of potential in SAVER to be applied to problems in mathematics and related fields, including the field of deep (reinforcement) learning itself.

Acknowledgments

Paper under double-blind review.

REFERENCES

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, David Silver, and Hado P. van Hasselt. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, 2017. URL http://papers.nips.cc/paper/6994-successor-features-for-transfer-in-reinforcement-learning.

André Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel J. Mankowitz, Augustin Zídek, and Rémi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, 2018. URL http://proceedings.mlr.press/v80/barreto18a.html.

Tim Brys, Ann Nowé, Daniel Kudenko, and Matthew E. Taylor. Combining multiple correlated reward and shaping signals by measuring confidence. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pp. 1687–1693, 2014. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8390.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 1104–1113, 2018a. URL http://proceedings.mlr.press/v80/dabney18a.html.

Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018b. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17184.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 1406–1415, 2018. URL http://proceedings.mlr.press/v80/espeholt18a.html.

Samuel J. Gershman, Bijan Pesaran, and Nathaniel D. Daw. Human reinforcement learning subdivides structured action spaces by learning effector-specific values. *Journal of Neuroscience*, 29(43):13524–13531, 2009. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.2469-09.2009. URL http://www.jneurosci.org/content/29/43/13524.

D. Ha and J. Schmidhuber. World models. 2018. doi: 10.5281/zenodo.1207631. URL https://worldmodels.github.io.

M. C. Jones. Estimating densities, quantiles, quantile densities and density quantiles. *Annals of the Institute of Statistical Mathematics*, 44(4):721–727, Dec 1992. ISSN 1572-9052. doi: 10.1007/BF00053400. URL https://doi.org/10.1007/BF00053400.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Eric R Klinkhammer. Learning in complex domains: Leveraging multiple rewards through alignment. 2018.

Roger Koenker. *Quantile Regression*. Econometric Society Monographs. Cambridge University Press, 2005. doi: 10.1017/CBO9780511754098.

Roger Koenker and Kevin Hallock. Quantile regression: An introduction. *Journal of Economic Perspectives*, 15(4):43–56, 2001.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

C. Liu, X. Xu, and D. Hu. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, March 2015. ISSN 2168-2216. doi: 10.1109/TSMC.2014.2358639.

Chen Ma, Junfeng Wen, and Yoshua Bengio. Universal successor representations for transfer reinforcement learning. *CoRR*, abs/1804.03758, 2018. URL http://arxiv.org/abs/1804.03758.

R. Andrew McCallum. Using transitional proximity for faster reinforcement learning. In Derek Sleeman and Peter Edwards (eds.), *Machine Learning Proceedings 1992*, pp. 316 – 321. Morgan Kaufmann, San Francisco (CA), 1992. ISBN 978-1-55860-247-2. doi: https://doi.org/10.1016/B978-1-55860-247-2.50045-0. URL http://www.sciencedirect.com/science/article/pii/B9781558602472500450.

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

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1928–1937, 2016. URL http://jmlr.org/proceedings/papers/v48/mniha16.html.

Hossam Mossalam, Yannis M Assael, Diederik M Roijers, and Shimon Whiteson. Multi-objective deep reinforcement learning. *arXiv preprint arXiv:1610.02707*, 2016.

Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29(2):429–443, 1997.

Thanh Thi Nguyen. A multi-objective deep reinforcement learning framework. *arXiv preprint arXiv:1803.02965*, 2018.

Georg Ostrovski, Will Dabney, and Rémi Munos. Autoregressive quantile networks for generative modeling. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 3933–3942, 2018. URL http://proceedings.mlr.press/v80/ostrovski18a.html.

Emanuel Parzen. Nonparametric statistical data modeling. *Journal of the American statistical association*, 74(365):105–121, 1979.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 2778–2787, 2017. URL http://proceedings.mlr.press/v70/pathak17a.html.

Daniel F. Cardozo Pinto and Stephan Lammel. Viral vector strategies for investigating midbrain dopamine circuits underlying motivated behaviors. *Pharmacology Biochemistry and Behavior*, 2017. ISSN 0091-3057. doi: https://doi.org/10.1016/j.pbb.2017.02.006. URL http://www.sciencedirect.com/science/article/pii/S0091305716303185.

Sébastien Racanière, Theophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 5694–5705, 2017. URL http://papers.nips.cc/paper/7152-imagination-augmented-agents-for-deep-reinforcement-learning.

Diederik M. Roijers and Shimon Whiteson. *Multi-Objective Decision Making*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2017. doi: 10.2200/S00765ED1V01Y201704AIM034. URL https://doi.org/10.2200/S00765ED1V01Y201704AIM034.

Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Intell. Res.*, 48:67–113, 2013. doi: 10.1613/jair.3987. URL https://doi.org/10.1613/jair.3987.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pp. 1057–1063, 1999.

Tomasz Tajmajer. Multi-objective deep q-learning with subsumption architecture. *arXiv preprint arXiv:1704.06676*, 2017.

John W Tukey. Which part of the sample contains the information? *Proceedings of the National Academy of Sciences*, 53(1):127–134, 1965.

Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5392–5402. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7123-hybrid-reward-architecture-for-reinforcement-learning.pdf.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017*, 2017. doi: 10.1109/IROS.2017.8206049. URL https://doi.org/10.1109/IROS.2017.8206049.

Table 1: List of hyperparameters used in all experiments

Hyperparameter	**Value**
Number of actors	16
Rollout length n	8
A2C entropy β	0.0
Critic λ_c	0.5
Pretrain learning rate	$0.1 \cdot$ Learning rate
Monotonic λ_{mon}	1.0
Discount factor γ	0.99
Quantile samples K	32
Optimizer	RMSProp
Non-linearities	ReLu

A ARCHITECTURE DETAILS

As agent network, we used a simple layer-wise fully connected network with 3 hidden representations of size 256, 128 and 64, respectively. From the last hidden representation we map with a linear layer to the action mean $\boldsymbol{\mu}$, while the action standard deviation $\boldsymbol{\sigma}$ is initialized to $e^{-1} \cdot [1, ..., 1]^T$ and kept independent of the observation input. For the A2C implementation, the last hidden representation of the agent network is also mapped through a linear layer to the scalar value estimate $V(\boldsymbol{o})$. For the PCPN we first map from $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and $\boldsymbol{o}$ to a hidden representation of size 256. From this hidden representation we map to d hidden representations, each of size 128, where d is the number of state dimensions. We then multiply each of these d hidden representation with a 128-dimensional cosine embedding of corresponding quantile target τ_j with $j \in \{1, ..., d\}$. Each of this new hidden representations is then fed through a fully connected layer with 64 neurons before being linearly projected to a scalar value representing the position change estimation of dimension j for quantile τ_j. We implement the QRRL critic as a separate network with 3 hidden representations of size 256, 128 and 64 and the same input as fed to the PCPN's first layer. Besides the action distribution representation which is ajusted corresponding to the experiment, we keep this architecture and all of the hyperparameters (as listed in Table 1) fixed for all experiments. The only hyperparameter we adjust is the learning rate, as described in the main text.