UNCERTAINTY-DRIVEN EXPLORATION FOR GENERALIZATION IN REINFORCEMENT LEARNING

Yiding Jiang*
Carnegie Mellon University
yidingji@cs.cmu.edu

J. Zico Kolter
Carnegie Mellon University
zkolter@cs.cmu.edu

Roberta Raileanu
Meta AI Research
raileanu@meta.com

ABSTRACT

Value-based methods are competitive when trained and tested in single environments. However, they fall short when trained on multiple environments with similar characteristics and tested on new ones from the same family. We investigate the potential reasons behind the poor generalization performance of value-based methods and discover that exploration plays a crucial role in these settings. Exploration is helpful not only for finding optimal solutions to the training environments but also for acquiring knowledge that helps generalization to other unseen environments. We show how to make value-based methods competitive in these settings by using uncertainty-driven exploration and distributional RL. Our algorithm is the first value-based approach to achieve state-of-the-art on both Procgen and Crafter, two challenging benchmarks for generalization in RL.

1 INTRODUCTION

Value-based methods (Watkins & Dayan, 1992) (which directly derive a policy from the value functions) tend to be competitive on *singleton* Markov decision processes (MDPs) where agents are trained and tested on the same environment (Mnih et al., 2013; Hessel et al., 2018; Badia et al., 2020). However, they fall short in *contextual* MDPs (CMDPs) (Wang et al., 2020; Ehrenberg et al., 2022), where agents are trained on a number of different environments that share a common structure and tested on unseen environments from the same family (Cobbe et al., 2019; Wang et al., 2020; Mohanty et al., 2021; Ehrenberg et al., 2022). In this work, we aim to understand *potential reasons for why value-based approaches work well in singleton MDPs but not in contextual MDPs and how we can make them competitive in CMDPs*.

Most of the existing approaches for improving generalization in CMDPs have treated the problem as a pure representation learning problem, applying regularization techniques which are commonly used in supervised deep learning (Farebrother et al., 2018; Cobbe et al., 2018; Igl et al., 2019; Lee et al., 2020; Ye et al., 2020; Laskin et al., 2020; Raileanu et al., 2020). However, these methods neglect the unique structure of reinforcement learning (RL), namely that agents collect their own data by exploring their environments. This suggests that there may be other avenues for improving generalization in RL beyond representation learning.

Here, we identify **the agent's exploration strategy as a key factor influencing generalization in contextual MDPs**. First, exploration can accelerate training in RL, and since neural networks may naturally generalize, better exploration can result in better training performance and consequently better generalization performance. Moreover, in singleton MDPs, exploration can only benefit decisions in that environment, while in CMDPs exploration in one environment can also help decisions in other, potentially unseen, environments. This is because learning about other parts of the environment can be useful in other MDPs even if it is not useful for the current MDP. As shown in Figure 1, trajectories that are suboptimal in certain MDPs may turn out to be optimal in other MDPs from the same family, so this knowledge can help find the optimal policy more quickly in a new MDP encountered during training, and better generalize to new MDPs without additional training.

One goal of exploration is to learn new things about the (knowable parts of the) environment so as to asymptotically reduce epistemic uncertainty. To model epistemic uncertainty (which is reducible

*Work done while interning at Meta AI Research.

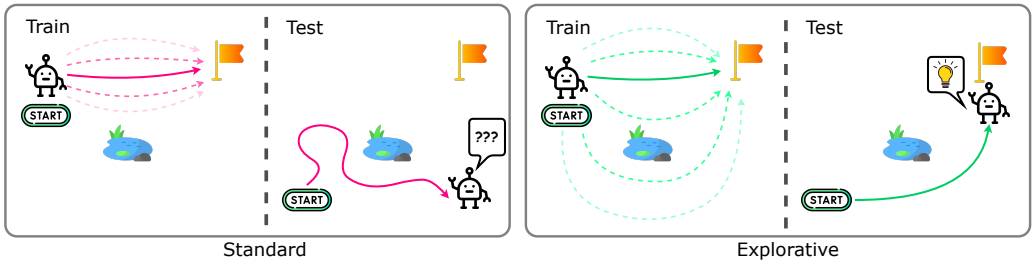


Figure 1: Exploration can help agents generalize to new situations at test time, even if it is not needed for finding the optimal policy on the training environments.

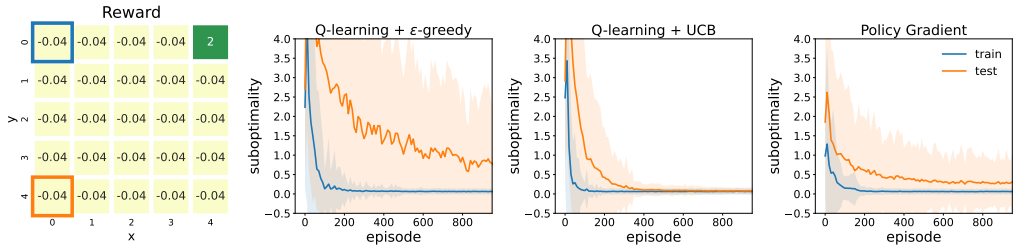
by acquiring more data), we need to disentangle it from aleatoric uncertainty (which is irreducible and stems from the inherent stochasticity of the environment). As first observed by Raileanu & Fergus (2021), in CMDPs the same state can have different values depending on the environment, but the agent does not know which environment it is in so it cannot perfectly predict the value of such states. This is a type of aleatoric uncertainty which can be modeled by learning a distribution over all possible values rather than a single point estimate (Bellemare et al., 2017). Based on these observations, we propose **Exploration via Distributional Ensemble** (**EDE**), a method that uses an ensemble of Q-value distributions to encourage exploring states with large epistemic uncertainty. We evaluate our approach on both Procgen (Cobbe et al., 2019) and Crafter (Hafner, 2022), two procedurally generated CMDP benchmarks for generalization in deep RL, demonstrating a significant improvement over more naïve exploration strategies. This is the first model-free value-based method to achieve state-of-the-art performance on these benchmarks, in terms of both sample efficiency and generalization, surpassing strong policy-optimization baselines (*i.e.,* methods that learn a parameterized policy in addition to a value function) and even a model-based one.

To summarize, this work: (i) identifies exploration as a key factor for generalization in CMDPs, (ii) supports this hypothesis using a didactic example in a tabular CMDP, (iii) proposes an exploration method based on minimizing the agent's epistemic uncertainty, and (iv) achieves state-of-the-art performance on two generalization benchmarks for deep RL, Procgen and Crafter.

2 Background

Episodic Reinforcement Learning. A *Markov decision process* (MDP) is defined by the tuple $\mu = (\mathcal{S}, \mathcal{A}, R, P, \gamma, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $R : \mathcal{S} \times \mathcal{A} \to [R_{\min}, R_{\max}]$ is the reward function , $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the transition distribution, $\gamma \in (0, 1]$ is the discount factor, and $\rho_0 : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the initial state distribution. We further denote the trajectory of an episode to be the sequence $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, r_0, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T, r_T, \boldsymbol{s}_{T+1})$ where $r_t = R(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and T is the length of the trajectory which can be infinite. If a trajectory is generated by a probabilistic policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$, $Z^\pi = \sum_{t=0}^{T} \gamma^t r_t$ is a random variable that describes the *discounted return* the policy achieves. The objective is to find a $\pi^\star$ that maximizes the expected discounted return, $\pi^\star = \arg\max_\pi \mathbb{E}_{\tau \sim p^\pi(\cdot)}[Z^\pi]$, where $p^\pi(\tau) = \rho_0(\boldsymbol{s}_0) \prod_{t=0}^{T} P(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t) \pi(\boldsymbol{s}_t \mid \boldsymbol{a}_t)$. For simplicity, we will use $\mathbb{E}_\pi$ instead of $\mathbb{E}_{\tau \sim p^\pi(\cdot)}$ to denote the expectation over trajectories sampled from the policy π. With a slight abuse of notation, we use $Z^\pi(\boldsymbol{s}, \boldsymbol{a})$ to denote the conditional discounted return when starting at $\boldsymbol{s}$ and taking action $\boldsymbol{a}$ (*i.e.,* $\boldsymbol{s}_0 = \boldsymbol{s}$ and $\boldsymbol{a}_0 = \boldsymbol{a}$). Finally, without loss of generality, we assume all measures are discrete and their values lie within $[0, 1]$.

Value-based methods (Watkins & Dayan, 1992) rely on a fundamental quantity in RL, the state-action value function, also referred to as the *Q-function*, $Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_\pi[Z^\pi \mid \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}]$. The Q-function of a policy can be found at the fixed point of the Bellman operator, $\mathcal{T}^\pi$ (Bellman, 1957), $\mathcal{T}^\pi Q(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\boldsymbol{s}' \sim P(\cdot \mid \boldsymbol{s}, \boldsymbol{a}),\, \boldsymbol{a}' \sim \pi(\cdot \mid \boldsymbol{s}')}[R(\boldsymbol{s}, \boldsymbol{a}) + \gamma Q(\boldsymbol{s}', \boldsymbol{a}')]$. Bellemare et al. (2017) extends the procedure to the distribution of discounted returns, $\mathcal{T}^\pi Z(\boldsymbol{s}, \boldsymbol{a}) \overset{d}{=} R(\boldsymbol{s}, \boldsymbol{a}) + \gamma Z(\boldsymbol{s}', \boldsymbol{a}')$, $\boldsymbol{s}' \sim P(\cdot \mid \boldsymbol{s}, \boldsymbol{a})$ and $\boldsymbol{a}' \sim \pi(\cdot \mid \boldsymbol{s}')$, where $\overset{d}{=}$ denotes that two random variables have the same distributions. This extension is referred to as *distributional RL* (we provide a more detailed description of QR-DQN, the distributional RL algorithm we use, in Appendix B.1). For value-based methods, the policy is directly derived from the Q-function as $\pi(\boldsymbol{a} \mid \boldsymbol{s}) = \mathbb{1}_{\arg\max_{\boldsymbol{a}'} Q(\boldsymbol{a}', \boldsymbol{s})}(\boldsymbol{a})$.



(a) Tabular CMDP (b) ε-greedy (c) UCB (d) Policy Gradient

Figure 2: (a) A tabular CMDP that illustrates the importance of exploration for generalization in RL. During training, the agent starts in the blue square, while at test time it starts in the orange square. In both cases, the goal is to get to the green square. The other plots show the mean and standard deviation of the train and test suboptimality (difference between optimal return and achieved return) over 100 runs for (b) Q-learning with ε-greedy exploration, (c) Q-learning with UCB exploration, and (d) policy gradient with entropy bonus.

Policy optimization approaches (Williams, 1992; Sutton et al., 1999) on the other hand seeks to directly learn the optimal policy via Monte Carlo estimation of the return. Specifically, given a policy π_ϑ parameterized by ϑ, we can compute the gradient of the expected return with respect to ϑ through the policy gradient theorem, $\nabla_\vartheta \mathbb{E}_{\pi_\vartheta}\left[Z^\pi\right] = \mathbb{E}_{\pi_\vartheta}\left[\sum_{t=0}^{T} \Psi(\boldsymbol{s}_t, \boldsymbol{a}_t) \nabla_\vartheta \log \pi_\vartheta(\boldsymbol{a}_t \mid \boldsymbol{s}_t)\right]$, where $\Psi(\boldsymbol{s}, \boldsymbol{a})$ is the *advantage function* (Schulman et al., 2015). ϑ is updated with gradient-based optimization method such as stochastic gradient descent. The final policy involves directly sampling from $\pi_\vartheta(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$. In general, the advantage function $\Psi(\boldsymbol{s}, \boldsymbol{a})$ can also involve learned value functions. However, the crucial feature of policy optimization methods is that they have a separate policy function, whereas value-based methods do not.

Generalization in Contextual MDPs. A *contextual Markov decision process* (CMDP) (Hallak et al., 2015) is a special class of *partially observable Markov decision process* (POMDP) consisting of different MDPs, that share state and action spaces but have different R, P, and ρ_0. In addition to the standard assumptions of a CMDP, we assume the existence of a structured distribution $q_\mu(\mu)$ over the MDPs. During training, we are given a (finite or infinite) number of training MDPs, $\mathcal{M}_{\text{train}} = \{\mu_1, \mu_2, \ldots, \mu_n\}$, drawn from q_μ (Ghosh et al. (2021) refers to this setting as epistemic POMDP). We use $p^{\pi,\mu}(\tau)$ to denote the trajectory distribution of rolling out π in μ. The objective is to find a single policy π that maximizes the expected discounted return over the entire distribution of MDPs, $\mathbb{E}_{\tau \sim p^{\pi,\mu}(\cdot), \mu \sim q_\mu(\cdot)}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$. Furthermore, we assume the existence of $\pi^\star$ (potentially more than one) that is optimal for all $\mu \in \text{supp}(q_\mu)$, since otherwise zero-shot generalization can be intractable (Malik et al., 2021). If the number of training environments is infinite, the challenge is learning good policies for all of them in a sample-efficient manner, *i.e.,* optimization; if it is finite, the challenge is also generalization to unseen environments.

3 A Didactic Example: Generalization in a Tabular CMDP

Much of the literature treats generalization in deep RL as a pure representation learning problem (Song et al., 2020; Zhang et al., 2020a;b; Agarwal et al., 2021a) and aims to improve it by using regularization (Farebrother et al., 2018; Zhang et al., 2018a; Cobbe et al., 2018; Igl et al., 2019) or data augmentation (Cobbe et al., 2018; Lee et al., 2020; Ye et al., 2020; Kostrikov et al., 2020; Laskin et al., 2020; Raileanu et al., 2020; Wang et al., 2020). In this section, we will show that the problem of generalization in RL extends beyond representation learning by considering a tabular CMDP which does not require representation learning. The goal is to provide intuition on the role of exploration for generalization in RL using a toy example. More specifically, we show that exploring the training environments can be helpful not only for finding rewards in those environments but also for making good decisions in new environments encountered at test time.

Concretely, we consider a generic 5×5 grid environment (Figure 2a). During training, the agent always starts at a fixed initial state, $(x = 0, y = 0)$, and can move in 4 cardinal directions (*i.e.,* up, down, left, right). The transition function is deterministic and if the agent moves against the boundary, it ends up at the same location. If the agent reaches the terminal state, $(x = 4, y = 0)$, it receives a large positive reward, $r = 2$ and the episode ends. Otherwise, the agent receives a small

negative reward, $r = -0.04$. At test time, the agent starts at a *different* location, $(x = 0, y = 4)$. In other words, the train and test MDPs only differ by their initial state distribution. In addition, each episode is terminated at 250 steps (10 times the size of the state space) to speed up the simulation, but most episodes reach the terminal state before forced termination.

We study 3 classes of algorithms with different exploration strategies: (1) Q-learning with ε-greedy (`Greedy`, Watkins & Dayan (1992)), (2) Q-learning with UCB (`UCB`, Auer et al. (2002)), and (3) policy gradient with entropy bonus (`PG`, Williams & Peng (1991)). To avoid any confounding effects of function approximation, we use *tabular* policy parameterization for Q-values and unnormalized log action probabilities. Both `Greedy` and `UCB` use the same base Q-learning algorithm (Watkins & Dayan, 1992). `Greedy` explores with ε-greedy strategy which takes a random action with probability ε and the best action according to the Q-function $\arg\max_a Q(s, a)$ with probability $1 - \varepsilon$. In contrast, `UCB` is uncertainty-driven so it explores more actions that have been selected fewer times in the past, according to $\pi(a \mid s) = \mathbb{1}(a = \arg\max_{a'} Q(s, a') + c\sqrt{\log(t)/N(s, a')})$, where t is the total number of timesteps, c is the exploration coefficient, and $N(s, a)$ is the number of times the agent has taken action a in state s, with ties broken randomly[1]. `PG` uses the policy gradient method with reward-to-go for variance reduction and entropy bonus for exploration. While `Greedy` is a naïve but widely used exploration strategy (Sutton & Barto, 2018), `UCB` is an effective algorithm designed for multi-armed bandits (Auer et al., 2002; Audibert et al., 2009). Building on their intuition, Chen et al. (2017) showed that uncertainty-based exploration bonus also performs well in challenging RL environments. See Appendix A for more details about the experimental setup.

Each method's exploration strategy is controlled by a single hyperparamter. For each hyperparameter, we search over 10 values and run every value for 100 trials. Each trial lasts for 1000 episodes. The results (mean and standard deviation) of hyperparameters with the highest average test returns for each method are shown in Figures 2b, 2c and 2d. We measure the performance of each method by its *suboptimality*, *i.e.,* the difference between the undiscounted return achieved by the learned policy and the undiscounted return of the optimal policy. While all three methods are able to quickly achieve optimal return for the training MDP, their performances on the test MDP differ drastically. First, we observe that `Greedy` has the worst generalization performance and the highest variance. On the other hand, `UCB` can reliably find the optimal policy for the test MDP as the final return has a negligible variance. Finally, `PG`'s performance lies between `Greedy` and `UCB`. On average, it converges to a better solution with lower variance than `Greedy` but it is not as good as `UCB`. In Appendix A.3, we provide another set of simulations based on changing dynamics and make similar observations. These experiments show that **more effective exploration of the training environments can result in better generalization to new environments**.

It is natural to ask whether this example is relevant for realistic deep RL problems. One hypothesis is that this tabular CMDP with two initial state distributions captures a common phenomenon in more challenging CMDPs like Procgen, namely that at test time, the agent can often end up in states that are suboptimal for the training MDPs. Having explored such states during training can help the agent recover from suboptimal states at test time. See Appendix I for an illustrative example inspired by one of the Procgen games. This is similar to covariate shift in imitation learning where the agent's suboptimality can compound over time. For the purpose of generalization, the effect of efficient exploration is, in spirit, similar to that of DAgger (Ross et al., 2011) — it helps the agent learn how to recover from suboptimal situations that are rare during training.

Another potential hypothesis for the superior performance of `UCB` relative to `Greedy` is that better exploration improves sample efficiency resulting in better training performance, which can in turn lead to better test performance. Indeed, even in the simple tabular MDPs from Figures 2 and 8, `UCB` converges faster on the training MDP. In Appendix E.1, we study this hypothesis in more detail. Note that the above two hypotheses can be simultaneously true, but the latter assumes the model learns generalizable representations, whereas the previous hypothesis applies regardless. In both cases, adapting uncertainty-driven exploration to deep RL on complex CMDPs has additional challenges such as representation learning and unobserved contexts. In the next section, we describe these challenges and propose a method to mitigate them.

[1]This `UCB` is a simple extension of the widely used bandits algorithm to MDP and does not enjoy the same regret guarantee, but we find it to be effective for didactic purpose. Azar et al. (2017) presents a formal but much more complicated extension of the UCB algorithm for value iteration on MDPs. The `UCB` used still represents uncertainty but is more similar to a count-based exploration bonus.

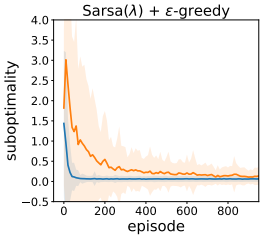


Figure 3: The results of Sarsa(λ).

Note that PG's softmax exploration strategy is not more sophisticated than ε-greedy so its performance gain over Greedy should be attributed to something else. We hypothesize that this improvement is due to the fact that algorithms based on policy gradient use whole-trajectory updates (*i.e.,* Monte Carlo methods) rather than one-step updates like Q-learning (*i.e.,* TD(0)). To test this hypothesis, we run Sarsa(λ) on the same environment. Sarsa(λ) (Sutton & Barto, 2018, Ch 12.7) is an on-policy method that learns a Q-value but also does whole-trajectory updates. We observe that with $\varepsilon = 0.9$ and $\lambda = 0.9$, Sarsa(λ) outperforms both Greedy and PG, but is still worse than UCB and has a much larger variance (Figure 3). This suggests that although whole-trajectory updates may partially be why policy gradient generalizes better than Q-learning, better exploration can improve generalization further. In the rest of the paper, we will focus on using exploration to improve generalization. Details and further analysis are in Appendix A.2.

4 Exploration via Distributional Ensemble

In the previous section, we showed we can improve generalization via exploration in the tabular setting. In this section, we would like to extend this idea to deep RL with function approximation. While in the tabular setting shown above there is no intrinsic stochasticity, environments can in general be stochastic (*e.g.,* random transitions, or random unobserved contexts). At a high level, epistemic uncertainty reflects a lack of knowledge which can be addressed by collecting more data, while aleatoric uncertainty reflects the intrinsic noise in the data which cannot be reduced regardless of how much data is collected. One goal of exploration can be to gather information about states with high epistemic uncertainty (O'Donoghue et al., 2018) since aleatoric uncertainty cannot be reduced, but typical estimates can contain both types of uncertainties (Kendall & Gal, 2017). In CMDPs, this is particularly important because a large source of aleatoric uncertainty is not knowing which context the agent is in (Raileanu & Fergus, 2021).

In this section, we introduce **Exploration via Distributional Ensemble** (**EDE**), a method that encourages the exploration of states with high epistemic uncertainty, which is computed using deep ensembles and distributional RL. While ensembles are a useful way of measuring uncertainty in neural networks (Lakshminarayanan et al., 2017), such estimates typically contain both epistemic and aleatoric uncertainty. Here, we build on Clements et al. (2019) who introduced an approach for disentangling the epistemic uncertainty from the aleatoric uncertainty of the learned Q-values.

Uncertainty Estimation. Clements et al. (2019) showed that learning the quantiles for QR-DQN (Dabney et al., 2018) can be formulated as a Bayesian inference problem, given access to a posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$, where $\boldsymbol{\theta}$ is the discretized quantiles of $Z(\boldsymbol{s}, \boldsymbol{a})$, and $\mathcal{D}$ is the dataset of experience on which the quantiles are estimated. Let $j \in [N]$ denote the index for the j^{th} quantile and $\mathcal{U}\{1, N\}$ denote the uniform distribution over integers between 1 and N. The uncertainty of the Q-value, $Q(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{j \sim \mathcal{U}\{1,N\}} [\boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a})]$, is the relevant quantity that can inform exploration. The overall uncertainty $\sigma^2 = \text{Var}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})} [Q(\boldsymbol{s}, \boldsymbol{a})]$ can be decomposed into *epistemic uncertainty* σ^2_{epi} and *aleatoric uncertainty* σ^2_{ale} such that $\sigma^2 = \sigma^2_{\text{epi}} + \sigma^2_{\text{ale}}$, where,

$$\sigma^2_{\text{epi}}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{j \sim \mathcal{U}\{1,N\}} \left[\text{Var}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})} [\boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a})] \right], \tag{1}$$

$$\sigma^2_{\text{ale}}(\boldsymbol{s}, \boldsymbol{a}) = \text{Var}_{j \sim \mathcal{U}\{1,N\}} \left[\mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})} [\boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a})] \right]. \tag{2}$$

Ideally, given an infinite or sufficiently large amount of diverse experience, one would expect the posterior to concentrate on the true quantile $\boldsymbol{\theta}^\star$, so $\text{Var}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\mathcal{D})} [\boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a})]$ and consequently $\sigma^2_{\text{epi}}(\boldsymbol{s}, \boldsymbol{a}) = 0$. σ^2_{ale} would be non-zero if the true quantiles have different values. Intuitively, to improve the sample efficiency, the agent should visit state-action pairs with high epistemic uncertainty in order to learn more about the environment (Chen et al., 2017). It should be noted that the majority of the literature on uncertainty estimation focuses on supervised learning; in RL, due to various factors such as bootstrapping, non-stationarity, limited model capacity, and approximate sampling, the uncertainty estimation generally contains errors but empirically even biased epistemic uncertainty is beneficial for exploration. We refer interested readers to Charpentier et al. (2022) for a more thorough discussion of this topic.

Sampling from $p(\boldsymbol{\theta} \mid \mathcal{D})$ is computationally intractable for complex MDPs and function approximators such as neural networks. Clements et al. (2019) approximates samples from $p(\boldsymbol{\theta} \mid \mathcal{D})$ with randomized MAP sampling (Pearce et al., 2020) which assumes a Gaussian prior over the model parameters. However, the unimodal nature of a Gaussian in parameter space may not have enough diversity for effective uncertainty estimation. Many works have demonstrated that *deep ensembles* tend to outperform regular ensembles and other approximate posterior sampling techniques (Lakshminarayanan et al., 2017; Fort et al., 2019) in supervised learning. Motivated by these observations, we propose to maintain M copies of fully-connected value heads, $g_i : \mathbb{R}^d \to \mathbb{R}^{|\mathcal{A}| \times N}$ that share a single feature extractor $f : \mathcal{S} \to \mathbb{R}^d$, similar to Osband et al. (2016a). However, we train each value head with *different* mini-batches and random initialization (*i.e.,* deep ensemble) instead of distinct data subsets (*i.e.,* bootstrapping). This is consistent with Lee et al. (2021) which shows deep ensembles usually perform better than bootstrapping for estimating uncertainty of Q-values but, unlike EDE, they do not decompose uncertainties.

Concretely, $i \in [M]$ is the index for M ensemble heads of the Q-network, and $j \in [N]$ is the index of the quantiles. The output of the i^{th} head for state $\boldsymbol{s}$ and action $\boldsymbol{a}$ is $\boldsymbol{\theta}_i(\boldsymbol{s}, \boldsymbol{a}) \in \mathbb{R}^N$, where the j^{th} coordinate, and $\boldsymbol{\theta}_{ij}(\boldsymbol{s}, \boldsymbol{a})$ is the j^{th} quantile of the predicted state-action value distribution for that head. The finite sample estimates of the two uncertainties are:

$$\hat{\sigma}^2_{\text{epi}}(\boldsymbol{s}, \boldsymbol{a}) = \frac{1}{N \cdot M} \sum_{j=1}^{N} \sum_{i=1}^{M} \left(\boldsymbol{\theta}_{ij}(\boldsymbol{s}, \boldsymbol{a}) - \bar{\boldsymbol{\theta}}_j(\boldsymbol{s}, \boldsymbol{a})\right)^2, \; \hat{\sigma}^2_{\text{ale}}(\boldsymbol{s}, \boldsymbol{a}) = \frac{1}{N} \sum_{j=1}^{N} \left(\bar{\boldsymbol{\theta}}_j(\boldsymbol{s}, \boldsymbol{a}) - Q(\boldsymbol{s}, \boldsymbol{a})\right)^2, \quad (3)$$

where $\bar{\boldsymbol{\theta}}_j(\boldsymbol{s}, \boldsymbol{a}) = \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{\theta}_{ij}(\boldsymbol{s}, \boldsymbol{a})$ and $Q(\boldsymbol{s}, \boldsymbol{a}) = \frac{1}{N} \sum_{j=1}^{N} \bar{\boldsymbol{\theta}}_j(\boldsymbol{s}, \boldsymbol{a})$.

Exploration Policy. There are two natural ways to use this uncertainty. The first one is by using Thompson sampling (Thompson, 1933) where the exploration policy is defined by sampling Q-values from the approximate posterior:

$$\pi_{\text{ts}}(\boldsymbol{a} \mid \boldsymbol{s}) = \mathbb{1}_{\arg\max_{\boldsymbol{a}'} \; \xi(\boldsymbol{s}, \boldsymbol{a}')}(\boldsymbol{a}), \text{ where } \quad \xi(\boldsymbol{s}, \boldsymbol{a}') \sim \mathcal{N}\left(Q(\boldsymbol{s}, \boldsymbol{a}'), \, \varphi \, \hat{\sigma}_{\text{epi}}(\boldsymbol{s}, \boldsymbol{a}')\right). \quad (4)$$

$\varphi \in \mathbb{R}_{\geq 0}$ is an exploration coefficient that controls how the agent balances exploration and exploitation. Alternatively, we can use the upper-confidence bound (UCB, Chen et al. (2017)):

$$\pi_{\text{ucb}}(\boldsymbol{a} \mid \boldsymbol{s}) = \mathbb{1}_{\boldsymbol{a}^\star}(\boldsymbol{a}), \text{ where } \boldsymbol{a}^\star = \arg\max_{\boldsymbol{a}'} \; Q(\boldsymbol{s}, \boldsymbol{a}') + \varphi \, \hat{\sigma}_{\text{epi}}(\boldsymbol{s}, \boldsymbol{a}') \quad (5)$$

which we found to achieve better results in CMDPs than Thompson sampling (used in Clements et al. (2019)) on Procgen when we use multiple parallel workers to collect experience, especially when combined with the next technique.

Equalized Exploration. Due to function approximation, the model may lose knowledge of some parts of the state space if it does not see them often enough. Even with UCB, this can still happen after the agent learns a good policy on the training environments. To increase the data diversity, we propose to use different exploration coefficients for each copy of the model used to collect experience. Concretely, we have K actors with synchronized weights; the k^{th} actor collects experience with the following policy:

$$\pi^{(k)}_{\text{ucb}}(\boldsymbol{a} \mid \boldsymbol{s}) = \mathbb{1}_{\boldsymbol{a}^\star}(\boldsymbol{a}), \text{ where } \boldsymbol{a}^\star = \arg\max_{\boldsymbol{a}'} \; Q(\boldsymbol{s}, \boldsymbol{a}') + \left(\varphi \, \lambda^{1 + \frac{k}{K-1}\alpha}\right) \hat{\sigma}_{\text{epi}}(\boldsymbol{s}, \boldsymbol{a}'). \quad (6)$$

$\lambda \in (0, 1)$ and $\alpha \in \mathbb{R}_{>0}$ are hyperparameters that control the shape of the coefficient distribution. We will refer to this technique as *temporally equalized exploration* (TEE). TEE is similar to Horgan et al. (2018) which uses different values of ε for the ε-greedy exploration for each actor. In practice, the performances are not sensitive to λ and α (see Figure 14 in Appendix E). Both learning and experience collection take place on a single machine and we do not use prioritized experience replay (Schaul et al., 2015) since prior work found it ineffective in CMDPs (Ehrenberg et al., 2022).

To summarize, our agent explores the environment in order to gather information about states with high epistemic uncertainty which is measured using ensembles and distributional RL. We build on the algorithm proposed in Clements et al. (2019) for estimating the epistemic uncertainty, but we use deep ensemble instead of MAP (Pearce et al., 2020), and use either UCB or Thompson sampling depending on the task and setting. In addition, for UCB, we propose that each actor uses a different exploration coefficient for more diverse data. While variations of the components of our algorithm have been used in prior works, this particular combination is new (see Appendix C). Our ablation experiments show that each design choice is important and naïvely combining these techniques does not work as well.

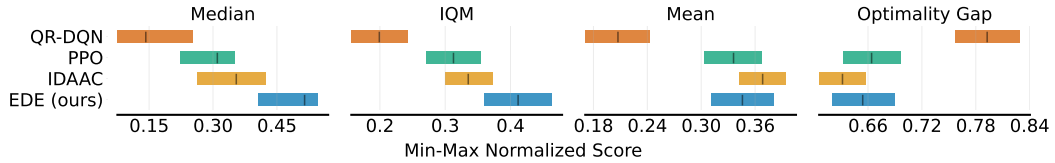


Figure 4: Test performance of different methods on the Procgen benchmark across 5 runs. Our method greatly outperforms all baselines in terms of median and IQM (the more statistically robust metrics), and is competitive with the state-of-the-art policy optimization methods in terms of mean and optimality gap. Optimality gap is equal to 1-mean for the evaluation configuration we chose.

5 Experiments

5.1 Procgen

We compare our method with 3 representative baselines on the standard Procgen benchmark (*i.e.,* 25M steps and easy mode) as suggested by Cobbe et al. (2018): (1) `QR-DQN` which is the prior state-of-the-art value-based method on Procgen (Ehrenberg et al., 2022); (2) `PPO` which is a popular policy optimization baseline on which most competitive methods are built; and (3) `IDAAC` which is state-of-the-art on Procgen and is built on `PPO`. We tune all hyperparameters of our method on the game `bigfish` only and evaluate all algorithms using the 4 metrics proposed in Agarwal et al. (2021b). We run each algorithm on every game for 5 seeds and report the aggregated min-max normalized scores on the full test distribution, and the estimated bootstrap-estimated 95% confidence interval in Figure 4 (simulated with the runs). Our approach significantly outperforms the other baselines in terms of median and interquartile mean (IQM) (which are the more statistically robust metrics according to Agarwal et al. (2021b)). In particular, it achieves almost 3 times the median score of `QR-DQN` and more than 2 times the IQM of `QR-DQN`. In terms of mean and optimality gap, our method is competitive with `IDAAC` and outperforms all the other baselines. To the best of our knowledge, this is the first value-based method that achieves such strong performance on Procgen. See Appendices G, D, and H for more details about our experiments, results, and hyperparameters.

Ablations. We aim to better understand how each component of our algorithm contributes to the final performance by running a number of ablations. In addition, we compare with other popular exploration techniques for value-based algorithms. Since many of the existing approaches are designed for DQN, we also adapt a subset of them to QR-DQN for a complete comparison. The points of comparison we use are: (1) Bootstrapped DQN (Osband & Van Roy, 2015) which uses bootstrapping to train several copies of models that have distinct exploration behaviors, (2) UCB (Chen et al., 2017) which uses an ensemble of models to do uncertainty estimation, (3) ϵz-greedy exploration (Dabney et al., 2021) which repeats the same random action (following a zeta distribution) to achieve temporally extended exploration, (4) UA-DQN (Clements et al., 2019), and (5) NoisyNet (Fortunato et al., 2017) which adds trainable noise to the linear layers. When UCB is combined with QR-DQN, we use the epistemic uncertainty unless specified otherwise. Details can be found in Appendix G.

First, note that both using the epistemic uncertainty via UCB and training on diverse data from TEE are crucial for the strong performance of EDE, with QR-DQN+TEE being worse than QR-DQN+UCB which is itself worse than EDE. Without using the epistemic uncertainty, the agent cannot do very well even if it trains on diverse data, *i.e.,* EDE is better than QR-DQN+TEE. Similarly, even if the agent uses epistemic uncertainty, it can still further improve its performance by training on diverse data, *i.e.,* EDE is better than QR-DQN+UCB.

Both Bootstrapped DQN and DQN+UCB, which minimize the total uncertainty rather than only the epistemic one, perform worse than DQN, although both of them are competitive exploration methods on Atari. This highlights the importance of using distributional RL in CMDPs in order to measure epistemic uncertainty. QR-DQN+UCB on the other hand does outperform QR-DQN and DQN because it seeks to reduce only the epistemic uncertainty. In Figure 12c from Appendix E, we show that indeed exploring with the total uncertainty σ^2 performs significantly worse at test time than exploring with only the epistemic uncertainty σ^2_{epi}. UA-DQN also performs worse than QR-DQN+UCB suggesting that deep ensemble may have better uncertainty estimation (Appendix F).

QR-DQN with ϵz-greedy exploration marginally improves upon the base QR-DQN, but remains significantly worse than our approach. This may be due to the fact that ϵz-greedy exploration can

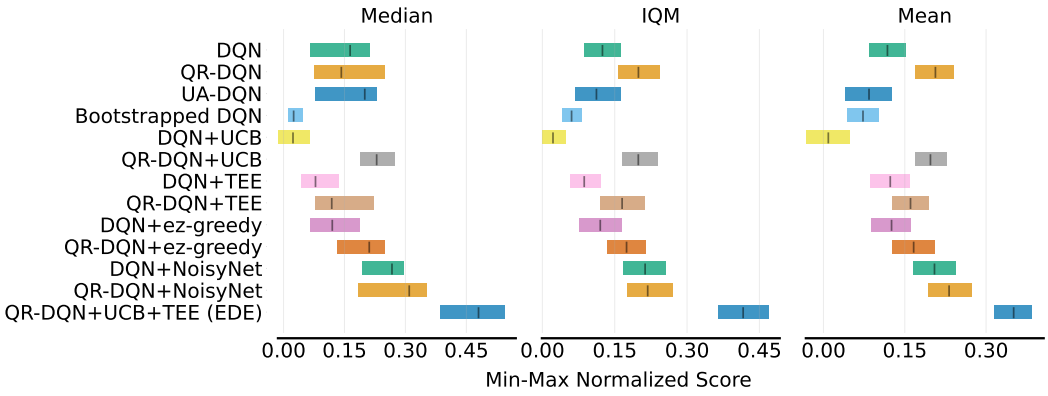


Figure 5: Test performance of different exploration methods on the Procgen benchmark across 5 runs.

induce temporally extended exploration but it is not aware of the agent's uncertainty. Not accounting for uncertainty can be detrimental since CMDPs can have a much larger number of effective states than singleton MDPs. If the models have already gathered enough information about a state, further exploring that state can hurt sample efficiency, regardless of whether the exploration is temporally extended or not.

NoisyNet performs better than the other points of comparison we consider, but it remains worse than EDE. Intuitively, the exploration behaviors of NoisyNet are naturally adaptive — the agents will take into account what they have already learned. While a direct theoretical comparison between NoisyNet and our method is hard to establish, we believe adaptivity is a common thread for methods that perform well on CMDPs. Nonetheless, if we consider IQM, none of these methods significantly outperforms one another whereas our method achieves a much higher IQM. Note that our TEE cannot be applied to NoisyNets and ϵz-greedy which already use an implicit "schedule".

5.2 CRAFTER

To test the generality of our method beyond the Procgen benchmark, we conduct experiments on the Crafter environment (Hafner, 2022). Making progress on Crafter requires a wide range of capabilities such as strong generalization, deep exploration, and long-term reasoning. Crafter evaluates agents using a score that summarizes the agent's abilities into a single number. Each episode is procedurally generated, so the number of training environments is practically infinite. While Crafter does not test generalization to new environments, it still requires generalization across the training environments in order to efficiently train on all of them. We build our method on top of the Rainbow implementation (Hessel et al., 2018) provided in the open-sourced code of Hafner (2022). We use Thompson sampling instead of UCB+TEE since only one environment is used. As seen in Table 5, our algorithm achieves significantly higher scores compared to all the baselines presented in Hafner (2022), including DreamerV2 (Hafner et al., 2021) which is a state-of-the-art model-based RL algorithm. The significant improvement over Rainbow, which is a competitive value-based approach, suggests that the exploration strategy is crucial for improving performance on such CMDPs.

Method	**Score (%)**
EDE (ours)	$\mathbf{11.7 \pm 1.0}$
Rainbow	4.3 ± 0.2
PPO	4.6 ± 0.3
DreamerV2	10.0 ± 1.2

Table 1: Results on Crafter after 1M steps and over 10 runs.

6 RELATED WORK

Generalization in RL. A large body of work has emphasized the challenges of training RL agents that can generalize to new environments and tasks (Rajeswaran et al., 2017; Machado et al., 2018; Justesen et al., 2018; Packer et al., 2018; Zhang et al., 2018a;b; Nichol et al., 2018; Cobbe et al., 2018; 2019; Juliani et al., 2019; Kuttler et al., 2020; Grigsby & Qi, 2020; Chen, 2020; Bengio et al., 2020; Bertrán et al., 2020; Ghosh et al., 2021; Kirk et al., 2021; Ajay et al., 2021; Ehrenberg et al., 2022; Lyle et al., 2022). This form of generalization is different from generalization in singleton

MDP which refers to function approximators generalizing to different states within the same MDP. A natural way to alleviate overfitting is to apply widely-used regularization techniques such as implicit regularization (Song et al., 2020), dropout (Igl et al., 2019), batch normalization (Farebrother et al., 2018), or data augmentation (Ye et al., 2020; Lee et al., 2020; Laskin et al., 2020; Raileanu et al., 2020; Wang et al., 2020; Yarats et al., 2021; Hansen & Wang, 2021; Hansen et al., 2021; Ko & Ok, 2022). Another family of methods aims to learn better state representations via bisimulation metrics (Zhang et al., 2020b;a; Agarwal et al., 2021a), information bottlenecks (Igl et al., 2019; Fan & Li, 2022), attention mechanisms (Carvalho et al., 2021), contrastive learning (Mazoure et al., 2021), adversarial learning (Roy & Konidaris, 2020; Fu et al., 2021; Rahman & Xue, 2021), or decoupling representation learning from decision making (Stooke et al., 2020; Sonar et al., 2020). Other approaches use information-theoretic approaches (Chen, 2020; Mazoure et al., 2020), non-stationarity reduction (Igl et al., 2020; Nikishin et al., 2022), curriculum learning (Jiang et al., 2020; Team et al., 2021; Jiang et al., 2021; Parker-Holder et al., 2022), planning (Anand et al., 2021), forward-backward representations (Touati & Ollivier, 2021), or diverse policies (Kumar et al., 2020). More similar to our work, Raileanu & Fergus (2021) show that the value function can overfit when trained on CMDPs and propose to decouple the policy from the value optimization to train more robust policies. However, this approach cannot be applied to value-based methods since the policy is directly defined by the Q-function. Most of the above works focus on policy optimization methods, and none emphasizes the key role exploration plays in training more general agents. In contrast, our goal is to understand why value-based methods are significantly worse on CMDPs.

Exploration. Exploration is a fundamental aspect of RL (Kolter & Ng, 2009; Fruit & Lazaric, 2017; Tarbouriech et al., 2020). Common approaches include ε-greedy (Sutton & Barto, 2018), count-based exploration (Strehl & Littman, 2008; Bellemare et al., 2016; Machado et al., 2020), curiosity-based exploration (Schmidhuber, 1991; Stanton & Clune, 2016; Pathak et al., 2017), or novelty-based methods specifically designed for exploring sparse reward CMDPs (Raileanu & Rocktäschel, 2020; Zha et al., 2021; Flet-Berliac et al., 2021; Zhang et al., 2021). These methods are based on policy optimization and focus on training agents in sparse reward CMDPs. In contrast, we are interested in leveraging exploration as a way of improving the generalization of value-based methods to new MDPs (including dense reward ones). Some of the most popular methods for improving exploration in value-based algorithms use noise (Osband et al., 2016b; Fortunato et al., 2017), bootstrapping (Osband & Van Roy, 2015; Osband et al., 2016a), ensembles (Chen et al., 2017; Lee et al., 2020; Liang et al., 2022), uncertainty estimation (Osband et al., 2013; O'Donoghue et al., 2018; Clements et al., 2019), or distributional RL (Mavrin et al., 2019). The main goal of these works was to use exploration to improve sample efficiency on singleton MDPs that require temporally extended exploration. Another related area is task-agnostic RL (Pong et al., 2019; Zhang et al., 2020c) where the agent explores the environment without reward and tries to learn a down-stream task with reward, but to our knowledge these methods have not been successfully adapted to standard benchmarks like Procgen where the MDPs can have large variations. Our work is the first one to highlight the role of exploration for faster training on contextual MDPs and better generalization to unseen MDPs. Our work also builds on the distributional RL perspective (Bellemare et al., 2017), which is useful in CMDPs for avoiding value overfitting (Raileanu & Fergus, 2021).

7 Discussion

In this work, we aim to understand why value-based methods are significantly worse than policy optimization methods in contextual MDPs, while being much better in singleton MDPs with similar characteristics. Our tabular experiments indicate that effective exploration is crucial for generalization to new environments. In CMDPs, exploring an environment is not only useful for finding the optimal policy in that environment, but also for acquiring knowledge that can be useful in other environments the agent may encounter at test time. In order to make value-based competitive with policy optimization in CDMPs, we propose an approach based on uncertainty-driven exploration and distributional RL. This is the first value-based based method to achieve state-of-the-art performance on both Procgen and Crafter, two challenging benchmarks for generalization in RL. Our experiments suggest that exploration is important for all RL algorithms trained and tested on CMDPs. While here we focus on value-based methods, similar ideas could be applied in policy optimization to further improve their generalization abilities. One limitation of our approach is that it is more computationally expensive due to the ensemble (Appendix F). Thus, we expect it could benefit from future advances in more efficient ways of accurately estimating uncertainty in neural networks.

8 Reproducibility Statement

We include the source code of EDE in the supplementary materials. This work builds on several existing open-source repositories and we have provided a detailed description of the algorithm, hyperparameters, and architecture in Appendix C and G. The codebase for Procgen is built on components from Jiang et al. (2020) and Ehrenberg et al. (2022); the codebase for Crafter is built on the baseline provided by Hafner (2022). Access to these codebases can all be found in these papers.

References

Rishabh Agarwal, Marlos C. Machado, P. S. Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. 2021a.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021b.

Anurag Ajay, Ge Yang, Ofir Nachum, and Pulkit Agrawal. Understanding the generalization gap in visual reinforcement learning. 2021.

Ankesh Anand, Jacob Walker, Yazhe Li, Eszter Vértes, Julian Schrittwieser, Sherjil Ozair, Théophane Weber, and Jessica B Hamrick. Procedural generalization by planning with self-supervised world models. *arXiv preprint arXiv:2111.01587*, 2021.

Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902, 2009.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *International Conference on Machine Learning*, pp. 263–272. PMLR, 2017.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pp. 507–517. PMLR, 2020.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458. PMLR, 2017.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. *ArXiv*, abs/2003.06350, 2020.

Martín Bertrán, N. Martínez, Mariano Phielipp, and G. Sapiro. Instance based generalization in reinforcement learning. *ArXiv*, abs/2011.01089, 2020.

Wilka Carvalho, Andrew Lampinen, Kyriacos Nikiforou, Felix Hill, and Murray Shanahan. Feature-attending recurrent modules for generalization in reinforcement learning. *arXiv preprint arXiv:2112.08369*, 2021.

Bertrand Charpentier, Ransalu Senanayake, Mykel Kochenderfer, and Stephan Günnemann. Disentangling epistemic and aleatoric uncertainty in reinforcement learning. *arXiv preprint arXiv:2206.01558*, 2022.

Jerry Zikun Chen. Reinforcement learning generalization with surprise minimization. *ArXiv*, abs/2004.12399, 2020.

Richard Y Chen, Szymon Sidor, Pieter Abbeel, and John Schulman. Ucb exploration via q-ensembles. *arXiv preprint arXiv:1706.01502*, 2017.

William R Clements, Bastien Van Delft, Benoît-Marie Robaglia, Reda Bahi Slaoui, and Sébastien Toth. Estimating risk and uncertainty in deep reinforcement learning. *arXiv preprint arXiv:1905.09638*, 2019.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*, 2018.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Will Dabney, Georg Ostrovski, and Andre Barreto. Temporally-extended ε-greedy exploration. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=ONBPHFZ7zG4.

Andy Ehrenberg, Robert Kirk, Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. A study of off-policy learning in environments with procedural content generation. In *ICLR Workshop on Agent Learning in Open-Endedness*, 2022.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Jiameng Fan and Wenchao Li. Dribo: Robust deep reinforcement learning via multi-view information bottleneck. In *International Conference on Machine Learning*, pp. 6074–6102. PMLR, 2022.

Jesse Farebrother, Marlos C. Machado, and Michael H. Bowling. Generalization and regularization in dqn. *ArXiv*, abs/1810.00123, 2018.

Yannis Flet-Berliac, Johan Ferret, Olivier Pietquin, Philippe Preux, and Matthieu Geist. Adversarially guided actor-critic. *CoRR*, abs/2102.04376, 2021. URL https://arxiv.org/abs/2102.04376.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Ronan Fruit and Alessandro Lazaric. Exploration-exploitation in mdps with options. In *Artificial intelligence and statistics*, pp. 576–584. PMLR, 2017.

Xiang Fu, Ge Yang, Pulkit Agrawal, and Tommi Jaakkola. Learning task informed abstractions. In *International Conference on Machine Learning*, pp. 3480–3491. PMLR, 2021.

Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability. *Advances in Neural Information Processing Systems*, 34:25502–25515, 2021.

Jake Grigsby and Yanjun Qi. Measuring visual generalization in continuous control from pixels. *ArXiv*, abs/2010.06740, 2020.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=1W0z96MFEoH.

Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13611–13617. IEEE, 2021.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Stabilizing deep q-learning with convnets and vision transformers under data augmentation. *Advances in Neural Information Processing Systems*, 34: 3680–3693, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Dy---0Z.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*, pp. 13956–13968, 2019.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Böhmer, and Shimon Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *ArXiv*, abs/2006.05826, 2020.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. *ArXiv*, abs/2010.03934, 2020.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021.

Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, J. Harper, Hunter Henry, Adam Crespi, J. Togelius, and D. Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *ArXiv*, abs/1902.01378, 2019.

Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv: Learning*, 2018.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

Byungchan Ko and Jungseul Ok. Efficient scheduling of data augmentation for deep reinforcement learning. *arXiv preprint arXiv:2206.00518*, 2022.

J Zico Kolter and Andrew Y Ng. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th annual international conference on machine learning*, pp. 513–520, 2009.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. One solution is not all you need: Few-shot extrapolation via structured maxent rl. *Advances in Neural Information Processing Systems*, 33:8198–8210, 2020.

Heinrich Kuttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *ArXiv*, abs/2006.13760, 2020.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations. https://openreview. net/forum*, 2020.

Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 6131–6141. PMLR, 2021.

Litian Liang, Yaosheng Xu, Stephen McAleer, Dailin Hu, Alexander Ihler, Pieter Abbeel, and Roy Fox. Reducing variance in temporal-difference value estimation via ensemble of deep networks. In *International Conference on Machine Learning*, pp. 13285–13301. PMLR, 2022.

Evan Z Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *International conference on machine learning*, pp. 6925–6935. PMLR, 2021.

Clare Lyle, Mark Rowland, Will Dabney, Marta Kwiatkowska, and Yarin Gal. Learning dynamics and generalization in reinforcement learning. *arXiv preprint arXiv:2206.02126*, 2022.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael H. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. In *IJCAI*, 2018.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5125–5133, 2020.

Dhruv Malik, Yuanzhi Li, and Pradeep Ravikumar. When is generalizable reinforcement learning tractable? *Advances in Neural Information Processing Systems*, 34:8032–8045, 2021.

Borislav Mavrin, Hengshuai Yao, Linglong Kong, Kaiwen Wu, and Yaoliang Yu. Distributional reinforcement learning for efficient exploration. In *International conference on machine learning*, pp. 4424–4434. PMLR, 2019.

Bogdan Mazoure, R'emi Tachet des Combes, Thang Doan, Philip Bachman, and R. Devon Hjelm. Deep reinforcement and infomax learning. *ArXiv*, abs/2006.07217, 2020.

Bogdan Mazoure, Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. Improving zero-shot generalization in offline reinforcement learning using generalized similarity functions. *arXiv preprint arXiv:2111.14629*, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Jonas Mockus. The bayesian approach to global optimization. In *System Modeling and Optimization*, pp. 473–481. Springer, 1982.

Sharada Mohanty, Jyotish Poonganam, Adrien Gaidon, Andrey Kolobov, Blake Wulfe, Dipam Chakraborty, Gražvydas Šemetulskis, João Schapke, Jonas Kubilius, Jurgis Pašukonis, et al. Measuring sample efficiency and generalization in reinforcement learning benchmarks: Neurips 2020 procgen benchmark. *arXiv preprint arXiv:2103.15332*, 2021.

Jun Morimoto and Kenji Doya. Robust reinforcement learning. *Neural computation*, 17(2):335–359, 2005.

Alex Nichol, V. Pfau, Christopher Hesse, O. Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *ArXiv*, abs/1804.03720, 2018.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 16828–16847. PMLR, 2022.

Ian Osband and Benjamin Van Roy. Bootstrapped thompson sampling and deep exploration. *arXiv preprint arXiv:1507.00300*, 2015.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. *Advances in Neural Information Processing Systems*, 26, 2013.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016a.

Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*, pp. 2377–2386. PMLR, 2016b.

Brendan O'Donoghue, Ian Osband, Remi Munos, and Volodymyr Mnih. The uncertainty bellman equation and exploration. In *International Conference on Machine Learning*, pp. 3836–3845, 2018.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Xiaodong Song. Assessing generalization in deep reinforcement learning. *ArXiv*, abs/1810.12282, 2018.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. *arXiv preprint arXiv:2203.01302*, 2022.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Tim Pearce, Felix Leibfried, and Alexandra Brintrup. Uncertainty in neural networks: Approximately bayesian ensembling. In *International conference on artificial intelligence and statistics*, pp. 234–244. PMLR, 2020.

Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Md Masudur Rahman and Yexiang Xue. Adversarial style transfer for robust policy optimization in reinforcement learning. 2021.

Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 8787–8798. PMLR, 2021.

Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkg-TJBFPB.

Roberta Raileanu, M. Goldstein, Denis Yarats, Ilya Kostrikov, and R. Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *ArXiv*, abs/2006.12862, 2020.

Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham M. Kakade. Towards generalization and simplicity in continuous control. *ArXiv*, abs/1703.02660, 2017.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Josh Roy and George Konidaris. Visual transfer for reinforcement learning via wasserstein domain confusion. *arXiv preprint arXiv:2006.03465*, 2020.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pp. 222–227, 1991.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Burr Settles. Active learning literature survey. 2009.

Anoopkumar Sonar, Vincent Pacelli, and Anirudha Majumdar. Invariant policy optimization: Towards stronger generalization in reinforcement learning. *ArXiv*, abs/2006.01096, 2020.

Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJli2hNKDH.

Christopher Stanton and Jeff Clune. Curiosity search: Producing generalists by encouraging individuals to continually explore and acquire skills throughout their lifetime. *PLOS ONE*, 11(9): 1–20, 09 2016. doi: 10.1371/journal.pone.0162235. URL https://doi.org/10.1371/journal.pone.0162235.

Adam Stooke, Kimin Lee, P. Abbeel, and M. Laskin. Decoupling representation learning from reinforcement learning. *ArXiv*, abs/2009.08319, 2020.

Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Jean Tarbouriech, Evrard Garcelon, Michal Valko, Matteo Pirotta, and Alessandro Lazaric. No-regret exploration in goal-oriented reinforcement learning. In *International Conference on Machine Learning*, pp. 9428–9437. PMLR, 2020.

Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

Ahmed Touati and Yann Ollivier. Learning one representation to optimize all rewards. *Advances in Neural Information Processing Systems*, 34:13–23, 2021.

K. Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. *ArXiv*, abs/2010.10814, 2020.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

Chang Ye, Ahmed Khalifa, Philip Bontrager, and Julian Togelius. Rotation, translation, and cropping for zero-shot generalization. *arXiv preprint arXiv:2001.09908*, 2020.

Daochen Zha, Wenye Ma, Lei Yuan, Xia Hu, and Ji Liu. Rank the episodes: A simple approach for exploration in procedurally-generated environments. *arXiv preprint arXiv:2101.08152*, 2021.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *ArXiv*, abs/1806.07937, 2018a.

Amy Zhang, Clare Lyle, Shagun Sodhani, Angelos Filos, Marta Kwiatkowska, Joelle Pineau, Yarin Gal, and Doina Precup. Invariant causal prediction for block mdps. *arXiv preprint arXiv:2003.06016*, 2020a.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020b.

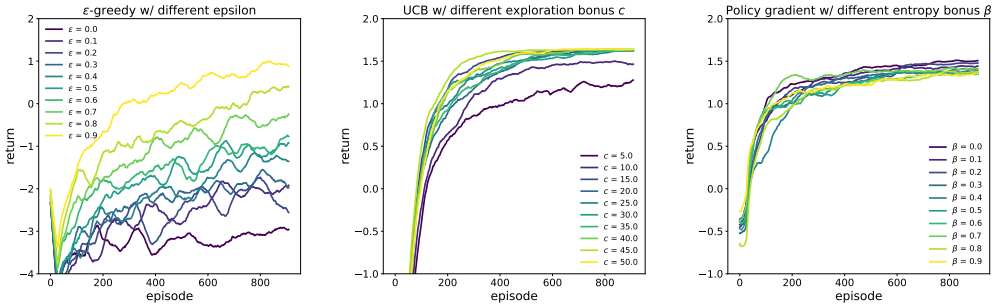
Chiyuan Zhang, Oriol Vinyals, Rémi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *ArXiv*, abs/1804.06893, 2018b.

Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E. Gonzalez, and Yuandong Tian. Noveld: A simple yet effective exploration criterion. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=CYUzpnOkFJp.

Xuezhou Zhang, Yuzhe Ma, and Adish Singla. Task-agnostic exploration in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:11734–11743, 2020c.

A TABULAR EXPERIMENTS

A.1 GENERALIZATION TO DIFFERENT INITIAL STATES



(a) ε-greedy (b) UCB (c) Policy Gradient

Figure 6: Mean test return for each method with different exploration hyperparameters.

Both the training MDP, μ_{train}, and the test MDP, μ_{test}, share the same state space $\mathcal{S}$, $[5] \times [5]$, and action space $\mathcal{A}$, $[4]$, corresponding to $\{\texttt{left}, \texttt{right}, \texttt{up}, \texttt{down}\}$. From each state, the transition, $P(s' \mid s, a)$, to the next state corresponding to an action is 100%. γ is 0.9. The two MDPs differ by only their initial state distribution: $\rho_{\text{train}}(s) = \mathbb{1}_{(0,0)}(s)$ and $\rho_{\text{test}}(s) = \mathbb{1}_{(4,0)}(s)$.

For both policy gradient and Q-learning, we use a base learning rate of $\alpha_0 = 0.05$ that decays as training progresses. Specifically, at time step t, $\alpha_t = \frac{1}{\sqrt{t}}\alpha_0$.

For Q-learning, the Q-function is parameterized as $\vartheta \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ where each entry is a state-action value. The update is:

$$Q^{(t+1)}(s_t, a_t) = Q^{(t)}(s_t, a_t) + \alpha_t \left(r_t + \gamma \max_a Q^{(t)}(s_{t+1}, a) - Q^{(t)}(s_t, a_t) \right), \tag{7}$$

where $Q(s, a) = \vartheta_{s,a}$. For Q-learning with UCB, the agent keeps $N(s, a)$ that keeps track of how many times the agent has taken action a from s over the course of training and explore according to:

$$\pi_{\text{ucb}}(a \mid s) = \mathbb{1}_{a^\star}(a) \;\text{ where }\; a^\star = \arg\max_a Q(s, a) + c\sqrt{\frac{\log t}{N(s, a)}}. \tag{8}$$

For Q-learning with ε-greedy, the exploration policy is:

$$\pi_{\text{egreedy}}(a \mid s) = (1 - \varepsilon)\, \mathbb{1}_{\arg\max_{a'} Q(s,a')}(a) + \varepsilon \left(\sum_{a' \in \mathcal{A}} \frac{1}{|\mathcal{A}|} \mathbb{1}_{a'}(a) \right). \tag{9}$$

Ties in $\arg\max$ are broken randomly which acts as the source of randomness for UCB.

For policy gradient, the policy is parameterized as $\vartheta \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ where each entry is the *logit* for the distribution of taking action a from s, *i.e.*, $\pi(a \mid s) = \frac{\exp(\vartheta_{s,a})}{\sum_{a' \in \mathcal{A}} \exp(\vartheta_{s,a'})}$. The overall gradient is:

$$\nabla_\vartheta \mathbb{E}_{\pi_\vartheta}[J(\tau)] = \mathbb{E}_{\pi_\vartheta}\left[\sum_{t=0}^{T} \nabla_\vartheta \log \pi_\vartheta(a_t \mid s_t) \sum_{k=t}^{T} \gamma^{k-t} r_k \right], \tag{10}$$

$$\nabla_\vartheta \mathcal{H}_{\pi_\vartheta} = \nabla_\vartheta \mathbb{E}_{\pi_\vartheta}\left[\sum_{t=0}^{T} \sum_{a \in \mathcal{A}} \pi_\vartheta(a \mid s_t) \log \pi_\vartheta(a \mid s_t) \right]. \tag{11}$$

The update is:

$$\vartheta^{(t+1)} = \vartheta^{(t)} + \alpha_t \left(\nabla_\vartheta \mathbb{E}_{\pi_\vartheta}[J(\tau)] + \beta \nabla_\vartheta \mathcal{H}_{\pi_\vartheta} \right)_{|\vartheta = \vartheta^{(t)}}, \tag{12}$$

where the expection is estimated with a single trajectory.

For each method, we repeat the experiment for the following 10 hyperparameter values:

- ε: $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$
- c: $\{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$
- β: $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$

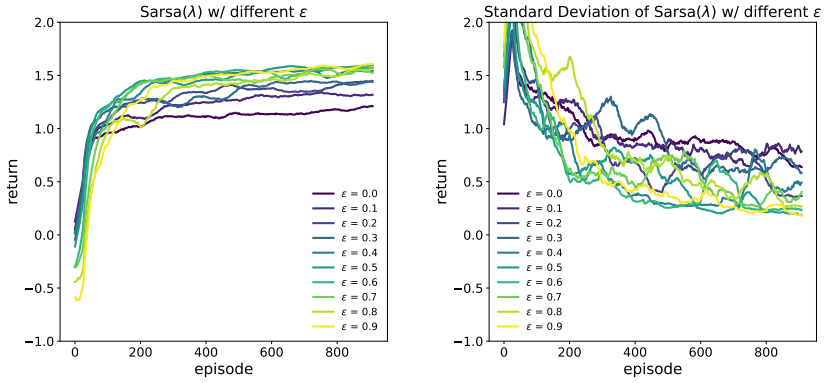
Each experiment involves training the algorithm for 100 trials. Each trial consists of 1000 episodes and each episode ends when the agent reaches a terminal state or at 250 steps. At test time, all approaches are deterministic: Q-learning follows the action with highest Q-value and policy gradient follows the action with the highest probability. The mean test returns for each experiment are shown in Figure 6. We observe that both UCB and policy gradient are relatively robust to the choice of hyperparameters, whereas ε-greedy's generalization performance varies greatly with ε. Furthermore, even with extremely large ε, the generalization performance is still far worse than UCB and policy gradient. The best performing values are respectively $\varepsilon^\star = 0.9$, $c^\star = 45$ and $\beta^\star = 0.1$.

A.2 SARSA(λ)

Sarsa(λ) is an on-policy value-based method that learns Q-value with whole-trajectory update via eligibility traces. Since the algorithmic details are more involved than the other methods we consider, we refer the reader to Chapter 12.7 of Sutton & Barto (2018) for the details. The Sarsa(λ) with ε-greedy experiments inherit all the hyperparameters of Q-learning + ε-greedy experiments, but have one more hyperparameter $\lambda \in (0, 1]$ that interpolates the method between Monte Carlo method and one-step TD update. We use a fixed $\lambda = 0.9$ for all experiments. We sweep ε across 10 values (100 trials each):

- ε: $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$

It can be seen from Figure 7a that even with whole-episode updates, the average return is still higher for larger exploration coefficients, which means exploration can still help. Furthermore, larger exploration coefficients also tend to have smaller variances in the test performance (Figure 7b).

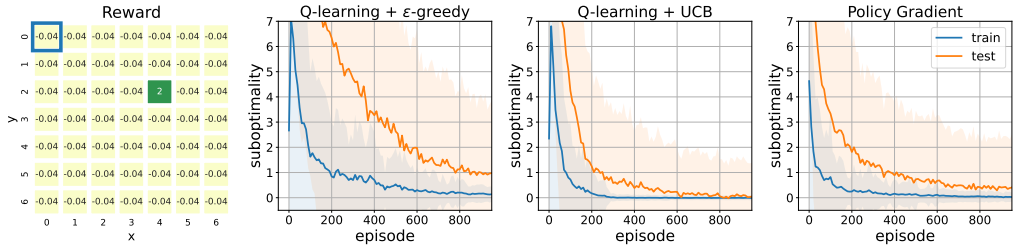


(a) Average of return. (b) Standard deviation of return.

Figure 7: Tet performance of Sarsa(λ) across different ε.

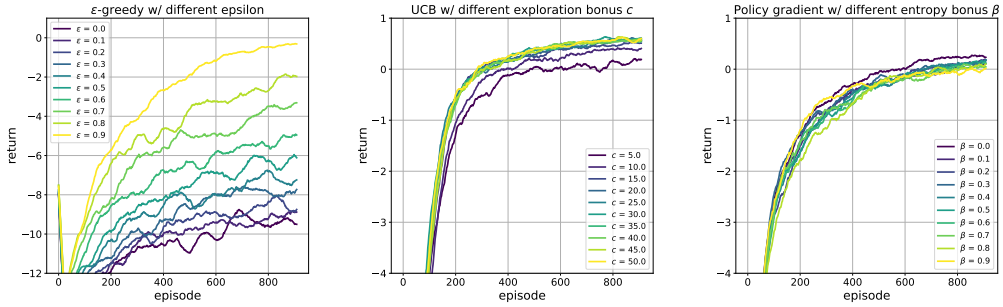
A.3 GENERALIZATION TO DIFFERENT DYNAMICS

In previous section, we presented a simple scenario that exploration helps generalization to different initial state distribution. Here, we demonstrate another scenario where exploration helps generalization to new dynamics. We use a 7×7 grid as the state space (Figure 8a). The agent receives a small negative reward for every location except for the green square $(4, 2)$, where the agent receives a reward of 2 and the episode ends. The difference between this environment and the previous one is that the agent always starts at $(0, 3)$ but at test time, a gust of wind can blow the agent off course. Specifically, with a probability of 40%, the agent will be blown to $+2$ unit in the y-direction. This environment is an example of the *windy world* from Sutton & Barto (2018). The results are shown in Figure 8 and 9, where we see similar results as the previous section, but at test time the optimal



(a) Tabular CMDP (b) ε-greedy (c) UCB (d) Policy Gradient

Figure 8: (a) During both training and test time, the agent starts in the blue square, but at test time, with 40% probability, wind can blow the agent down by two unit. In both cases, the goal is to get to the green square. The other plots show the mean and standard deviation of the train and test suboptimality (difference between optimal return and achieved return) over 100 runs for (b) Q-learning with ε-greedy exploration, (c) Q-learning with UCB exploration, and (d) policy gradient with entropy bonus.



(a) ε-greedy (b) UCB (c) Policy Gradient

Figure 9: Mean test return for each method with different exploration hyperparameters when the transition dynamic changes at test time.

expected return is generally worse and has higher variance for all methods since there is stochasticity (suboptimality is computed with respect to the return of an oracle that consistently moves towards the goal). This setup is related to *robust reinforcement learning* (Morimoto & Doya, 2005), but exploration makes no assumptions about the type or magnitude of perturbation (thus making more complicated perturbation, *e.g.,* different random initializations of the same game, possible) and does not require access to a simulator.

B Additional Background

B.1 Quantile Regression DQN

Quantile regression DQN (QR-DQN) is a variant of distributional RL that tries to learn the *quantiles* of the state-action value distribution with quantile regression rather than produce the actual distribution of Q-values. Compared to the original C51 (Bellemare et al., 2017), QR-DQN has many favorable theoretical properties. Here, we provide the necessary background for understanding QR-DQN and refer the readers to Dabney et al. (2018) for a more in-depth discussion of QR-DQN.

Given a random variable Y with associated CDF $F_Y(y) = \mathbb{P}(Y \leq y)$, the inverse CDF of Y is

$$F_Y^{-1}(\tau) = \inf \{y \in \mathbb{R} \mid \tau \leq F_Y(y)\}.$$

To approximate Y, we may discretize the CDF with N qunatile values:

$$\left\{F_Y^{-1}(\tau_1), F_Y^{-1}(\tau_2), \ldots, F_Y^{-1}(\tau_N)\right\} \quad \text{where} \quad \tau_i = \frac{i}{N}.$$

Using the same notation as main text, we use $\boldsymbol{\theta} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^N$ to denote a parametric function that maps a state-action pair, $(\boldsymbol{s}, \boldsymbol{a})$, to the estimated quantiles $\{\boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a})\}_{j=1}^N$. The approximated CDF

of $Z(\boldsymbol{s}, \boldsymbol{a})$ is thus:

$$\widehat{Z}(\boldsymbol{s}, \boldsymbol{a}) \stackrel{d}{=} \frac{1}{N} \sum_{j=1}^{N} \delta\left(\boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a})\right),$$

where $\delta\left(\boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a})\right)$ is the Dirac delta distribution centered at $\boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a})$. To learn the quantiles, we use *quantile regression* which allows for unbiased estimate of the true quantile values. Concretely, for a target distribution Z and quantile τ, the value of $F_Z^{-1}(\tau)$ is the minimizer of the quantile regression loss:

$$F_Z^{-1}(\tau) = \arg\min_{\theta} \mathcal{L}_{\text{QR}}^{\tau}(\theta) = \mathbb{E}_{z \sim Z}\left[\rho_\tau(z - \theta)\right] \quad \text{where} \quad \rho_\tau(u) = u\left(\tau - \mathbb{1}(u < 0)\right).$$

Due to discretization, for each quantile τ_j, the minimizer of the 1-Wasserstein distance within that quantile is at θ that satisfies $F_Z(\theta) = \frac{1}{2}(\tau_{j-1} + \tau_j) = \hat{\tau}_j$ (see Lemma 2 of Dabney et al. (2018)). Consequently, to approximate Z, we can simultaneously optimize for all of the quantiles, $\{\theta_j\}_{j=1}^{N}$, by minimizing the following loss:

$$\sum_{j=1}^{N} \mathcal{L}_{\text{QR}}^{\hat{\tau}_j}(\theta_j) = \sum_{j=1}^{N} \mathbb{E}_{z \sim Z}\left[\rho_{\hat{\tau}_j}(z - \theta_j)\right].$$

To stablize optimization, QR-DQN uses an modified version of quantile loss called *quantile Huber loss* which utilizes the Huber loss with hyperparameter κ:

$$\mathcal{L}_\kappa(u) = \frac{1}{2}u^2 \mathbb{1}(|u| \leq \kappa) + \kappa\left(|u| - \frac{1}{2}\kappa\right)\mathbb{1}(|u| > \kappa).$$

The quantile Huber loss is defined as:

$$\rho_\tau^\kappa(u) = \left|\tau - \mathbb{1}(u < 0)\right| \mathcal{L}_\kappa(u).$$

In QR-DQN, both the approximating distribution and the target distribution are discretized with quantiles. The target distribution is computed with bootstrapping through the distributional Bellman operator. Concretely, given a target network, $\boldsymbol{\theta}^{\text{target}}$, and transition tuple $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$, the target distribution is approximated as:

$$\begin{aligned} Q(\boldsymbol{s}', \boldsymbol{a}') &= \frac{1}{N} \sum_{j=1}^{N} \boldsymbol{\theta}_j^{\text{target}}(\boldsymbol{s}', \boldsymbol{a}') \\ \boldsymbol{a}^\star &= \arg\max_{\boldsymbol{a}'} Q(\boldsymbol{s}', \boldsymbol{a}') \\ \mathcal{T}\boldsymbol{\theta}_j &= r + \gamma \boldsymbol{\theta}_j^{\text{target}}(\boldsymbol{s}', \boldsymbol{a}^\star). \end{aligned} \tag{13}$$

The TD target Q-value distribution's quantiles and distribution are:

$$\{\mathcal{T}\boldsymbol{\theta}_1, \mathcal{T}\boldsymbol{\theta}_2, \ldots, \mathcal{T}\boldsymbol{\theta}_N\}, \quad \mathcal{T}Z \stackrel{d}{=} \frac{1}{N} \sum_{j=1}^{N} \delta\left(\mathcal{T}\boldsymbol{\theta}_j\right).$$

Treating the target distribution as an oracle (*i.e.,* not differentiable), we update the current quantile estimates to minimize the following quantile regression loss:

$$\sum_{j=1}^{N} \mathbb{E}_{z \sim \mathcal{T}Z}\left[\rho_{\hat{\tau}_j}^\kappa(z - \boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a}))\right] = \frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{N} \rho_{\hat{\tau}_j}^\kappa\left(\texttt{sg}\left[\mathcal{T}\boldsymbol{\theta}_i\right] - \boldsymbol{\theta}_j(\boldsymbol{s}, \boldsymbol{a})\right). \tag{14}$$

`sg` denotes the stop gradient operation as we do not optimize the target network (standard DQN practice).

C ALGORITHMIC DETAILS

In this section, we describe EDE in detail and highlight its differences from previous approaches. In Algorithm 1, we describe the main training loop of EDE. For simplicity, the pseudocode omits the initial data collection steps during which the agent uses a uniformly random policy to collect data and does not update the parameters. We also write the parallel experience collection as a for loop for easier illustration, but in the actual implementation, all loops over K are parallelized. The main algorithmic loop largely resembles the standard DQN training loop. Most algorithmic difference happen in how exploration is being performance *i.e.,* how the action is chosen and how the ensemble members are updated.

In Algorithm 2, we describe how EDE chooses exploratory actions during training. For each state, we first extract the features with the shared feature extractor f and then compute the state-action value distributions for each ensemble head. With the state-action value distributions, we compute the estimate for the epistemic uncertainty as well as the expected Q-values. Using these quantities, we choose the action based on either Thompson sampling or UCB.

In Algorithm 3, we describe how EDE updates the parameters of the feature extractor and ensemble heads. This is where our algorithm deviates significantly from prior approaches. For each ensemble head, we sample an independent minibatch from the replay buffer and compute the loss like *deep ensemble*. Inside the loop, the loss is computed as if each ensemble member is a QR-DQN. However, doing so naïvely means that the feature extractor f will be updated at a much faster rate. To ensure that all paramters are updated at the same rate, for each update, we randomly choose a single ensemble member that will be responsible for updating the feature extractor and the gradient of other ensemble heads are not propagated to the feature extractor. This also saves some compute. In practice, this turns out to be crucial for performing good uncertainty estimation and generalization. We show the effect of doing so in Figure 13c.

Attentive readers would notice EDE is more expensive than other methods since it needs to process M minibatches for each update. In our experiments, the speed of EDE with 5 ensemble member is approximately 2.4 times slower than the base QR-DQN (see Appendix F). On the other hand, we empirically observed that the deep ensemble is crucial for performing accurate uncertainty estimation in CMDPs. UA-DQN (Clements et al., 2019), which uses MAP sampling, performs much worse than EDE, even when both use the same number of ensemble members *e.g.,* 3 (see Figure 5). In this work, we do not investigate other altrenatives of uncertainty estimation but we believe future works in improving uncertainty estimation in deep learning could significantly reduce the computation overhead of EDE.

Algorithm 1 EDE

Initialize feature extractor f and ensemble heads $\{g_i\}_{i=1}^{M}$
Initialize target feature extractor f^{target} and ensemble heads $\{g_i^{\text{target}}\}_{i=1}^{M}$
Initialize replay buffer `Buffer`
$\{\boldsymbol{s}^{(k)}, \texttt{done}^{(k)}\}_{k=1}^{K} \leftarrow$ reset all environment
for t from 1 to T **do** ▷ Standard DQN training loop
  for k from 1 to K **do**
    $\boldsymbol{a} \rightarrow \textsc{ChooseAction}\big(\boldsymbol{s}^{(k)},\ f,\ \{g_i\}_{i=1}^{M}, k\big)$
    $\boldsymbol{s}', r, \texttt{done}^{(k)} \leftarrow$ Take $\boldsymbol{a}$ in the environment
    add $\{\boldsymbol{s}^{(k)}, \boldsymbol{a}, r, \boldsymbol{s}', \texttt{done}^{(k)}\}$ to `Buffer`
    $\boldsymbol{s}^{(k)} \leftarrow \boldsymbol{s}'$
    if $\texttt{done}^{(k)}$ **then**
      $\boldsymbol{s}^{(k)}, \texttt{done}^{(k)} \leftarrow$ reset the k^{th} environment
    end if
  end for
  $\textsc{Update}\big(\texttt{Buffer},\ f,\ \{g_i\}_{i=1}^{M},\ f^{\text{target}},\ \{g_i^{\text{target}}\}_{i=1}^{M}\big)$
  if t mod `update_frequency` = 0 **then**
    $f^{\text{target}},\ \{g_i^{\text{target}}\}_{i=1}^{M} \leftarrow f,\ \{g_i\}_{i=1}^{M}$
  end if
end for

Algorithm 2 CHOOSEACTION$\big(\boldsymbol{s},\ f,\ \{g_i\}_{i=1}^{M}, k\big)$

$\texttt{feature} \leftarrow f(\boldsymbol{s})$ ▷ Only one forward pass on the feature extractor
for i from 1 to M **do**
  $\{\boldsymbol{\theta}_{ij}(\boldsymbol{s}, \boldsymbol{a})\}_{j=1}^{N} \leftarrow g_i(\texttt{feature})$ ▷ Compute individual ensemble prediction
end for
Compute $\hat{\sigma}^2_{\text{epi}}(\boldsymbol{s}, \boldsymbol{a})$ using Equation 3 ▷ Estimate epistemic uncertainty with ensemble
Compute $\boldsymbol{a}^\star$ using Equation 6 or Equation 4 ▷ Explore with estimated uncertainty
return $\boldsymbol{a}^\star$

Algorithm 3 UPDATE$\big(\texttt{Buffer},\ f,\ \{g_i\}_{i=1}^{M},\ f^{\text{target}},\ \{g_i^{\text{target}}\}_{i=1}^{M}\big)$

$\mathcal{L} \leftarrow 0$
$\texttt{grad\_index} \sim \mathcal{U}\{1, M\}$ ▷ Randomly select an ensemble member
for i from 1 to M **do** ▷ Train each ensemble member independently
  $\{\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}', \texttt{done}\} \leftarrow$ Sample from $\texttt{Buffer}$
  $\texttt{feature}(s') \leftarrow f^{\text{target}}(\boldsymbol{s}')$
  $\{\boldsymbol{\theta}^{\text{target}}_{ij}(\boldsymbol{s}', \boldsymbol{a})\}_{j=1}^{N} \leftarrow g_i^{\text{target}}(\texttt{feature}(\boldsymbol{s}'))$
  Compute $\{\mathcal{T}\boldsymbol{\theta}_{i1}, \mathcal{T}\boldsymbol{\theta}_{i2}, \ldots, \mathcal{T}\boldsymbol{\theta}_{iN}\}$ using Equation 13 for the i^{th} ensemble member
  $\texttt{feature}(\boldsymbol{s}) \leftarrow f(\boldsymbol{s})$
  if $i \neq \texttt{grad\_index}$ **then** ▷ Prevent over-training the feature extractor
    $\texttt{feature}(\boldsymbol{s}) \leftarrow$ STOPGRADIENT $(\texttt{feature}(\boldsymbol{s}))$
  end if
  $\{\boldsymbol{\theta}_{ij}(\boldsymbol{s}, \boldsymbol{a})\}_{j=1}^{N} \leftarrow g_i(\texttt{feature}(\boldsymbol{s}))$
  $\mathcal{L}_i \leftarrow$ compute QR loss using Equation 14 for the i^{th} ensemble member
  $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_i$
end for
Compute gradient of $\mathcal{L}$ w.r.t the parameters of $f,\ \{g_i\}_{i=1}^{M}$
Update the parameters with Adam

D PROCGEN RESULTS

In Table 3 and Table 2, we show the mean and standard deviation of the final unnormalized train and test scores (at the end of 25M environment steps) of EDE along side with other methods in the literature. We adapt the tables from Raileanu & Fergus (2021). In Figure 10, we show the curves of min-max normalized test score that we reproduced using the code from Raileanu & Fergus (2021). We see that for a subset of games, EDE is significantly more sample-efficient. For other games, EDE is either on par with the policy optimization methods or fails to train, indicating there are other issues behind its poor performance on these games. Note that IDAAC results tune the hyperparameters for individual games whereas we only tune the hyperparameters on `bigfish`.

Game	PPO	MixReg	PLR	UCB-DRAC	PPG	DAAC	IDAAC	EDE
bigfish	3.7 ± 1.3	7.1 ± 1.6	10.9 ± 2.8	9.2 ± 2.0	11.2 ± 1.4	17.8 ± 1.4	18.5 ± 1.2	$\mathbf{22.1 \pm 2.0}$
StarPilot	24.9 ± 1.0	32.4 ± 1.5	27.9 ± 4.4	30.0 ± 1.3	47.2 ± 1.6	36.4 ± 2.8	37.0 ± 2.3	$\mathbf{49.6 \pm 2.4}$
FruitBot	26.2 ± 1.2	27.3 ± 0.8	28.0 ± 1.4	27.6 ± 0.4	27.8 ± 0.6	$\mathbf{28.6 \pm 0.6}$	27.9 ± 0.5	25.7 ± 1.4
BossFight	7.4 ± 0.4	8.2 ± 0.7	8.9 ± 0.4	7.8 ± 0.6	$\mathbf{10.3 \pm 0.2}$	9.6 ± 0.5	9.8 ± 0.6	10.0 ± 0.5
Ninja	5.9 ± 0.2	6.8 ± 0.5	$\mathbf{7.2 \pm 0.4}$	6.6 ± 0.4	6.6 ± 0.1	6.8 ± 0.4	6.8 ± 0.4	6.1 ± 0.6
Plunder	5.2 ± 0.6	5.9 ± 0.5	8.7 ± 2.2	8.3 ± 1.1	14.3 ± 2.0	20.7 ± 3.3	$\mathbf{23.3 \pm 1.4}$	4.9 ± 0.5
CaveFlyer	5.1 ± 0.4	6.1 ± 0.6	6.3 ± 0.5	5.0 ± 0.8	7.0 ± 0.4	4.6 ± 0.2	5.0 ± 0.6	$\mathbf{7.9 \pm 0.4}$
CoinRun	8.6 ± 0.2	8.6 ± 0.3	8.8 ± 0.5	8.6 ± 0.2	8.9 ± 0.1	9.2 ± 0.2	$\mathbf{9.4 \pm 0.1}$	6.7 ± 0.5
Jumper	5.9 ± 0.2	6.0 ± 0.3	5.8 ± 0.5	6.2 ± 0.3	5.9 ± 0.1	$\mathbf{6.5 \pm 0.4}$	6.3 ± 0.2	5.7 ± 0.3
Chaser	3.5 ± 0.9	5.8 ± 1.1	6.9 ± 1.2	6.3 ± 0.6	$\mathbf{9.8 \pm 0.5}$	6.6 ± 1.2	6.8 ± 1.0	1.6 ± 0.1
Climber	5.6 ± 0.5	6.9 ± 0.7	6.3 ± 0.8	6.3 ± 0.6	2.8 ± 0.4	7.8 ± 0.2	$\mathbf{8.3 \pm 0.4}$	5.7 ± 1.1
Dodgeball	1.6 ± 0.1	1.7 ± 0.4	1.8 ± 0.5	4.2 ± 0.9	2.3 ± 0.3	3.3 ± 0.5	3.2 ± 0.3	$\mathbf{13.3 \pm 0.1}$
Heist	2.5 ± 0.6	2.6 ± 0.4	2.9 ± 0.5	3.5 ± 0.4	2.8 ± 0.4	3.3 ± 0.2	$\mathbf{3.5 \pm 0.2}$	1.5 ± 0.3
Leaper	4.9 ± 2.2	5.3 ± 1.1	6.8 ± 1.2	4.8 ± 0.9	$\mathbf{8.5 \pm 1.0}$	7.3 ± 1.1	7.7 ± 1.0	6.4 ± 0.3
Maze	5.5 ± 0.3	5.2 ± 0.5	5.5 ± 0.8	$\mathbf{6.3 \pm 0.1}$	5.1 ± 0.3	5.5 ± 0.2	5.6 ± 0.3	3.4 ± 0.5
Miner	8.4 ± 0.7	9.4 ± 0.4	9.6 ± 0.6	9.2 ± 0.6	7.4 ± 0.2	8.6 ± 0.9	$\mathbf{9.5 \pm 0.4}$	0.7 ± 0.1

Table 2: Procgen scores on test levels after training on 25M environment steps. The mean and standard deviation are computed using 5 runs with different seeds.

Game	PPO	MixReg	PLR	UCB-DRAC	PPG	DAAC	IDAAC	EDE
bigfish	9.2 ± 2.7	15.0 ± 1.3	7.8 ± 1.0	12.8 ± 1.8	19.9 ± 1.7	20.1 ± 1.6	21.8 ± 1.8	$\mathbf{27.5 \pm 2.0}$
StarPilot	29.0 ± 1.1	28.7 ± 1.1	2.6 ± 0.3	33.1 ± 1.3	$\mathbf{49.6 \pm 2.1}$	38.0 ± 2.6	38.6 ± 2.2	46.9 ± 0.7
FruitBot	28.8 ± 0.6	29.9 ± 0.5	15.9 ± 1.3	29.3 ± 0.5	$\mathbf{31.1 \pm 0.5}$	29.7 ± 0.4	29.1 ± 0.7	27.7 ± 1.0
BossFight	8.0 ± 0.4	7.9 ± 0.8	8.7 ± 0.7	8.1 ± 0.4	$\mathbf{11.1 \pm 0.1}$	10.0 ± 0.4	10.4 ± 0.4	10.6 ± 0.4
Ninja	7.3 ± 0.3	8.2 ± 0.4	5.4 ± 0.5	8.0 ± 0.4	8.9 ± 0.2	8.8 ± 0.2	8.9 ± 0.3	$\mathbf{9.1 \pm 0.4}$
Plunder	6.1 ± 0.8	6.2 ± 0.3	4.1 ± 1.3	10.2 ± 1.76	16.4 ± 1.9	22.5 ± 2.8	$\mathbf{24.6 \pm 1.6}$	8.2 ± 0.8
CaveFlyer	6.7 ± 0.6	6.2 ± 0.7	6.4 ± 0.1	5.8 ± 0.9	9.5 ± 0.2	5.8 ± 0.4	6.2 ± 0.6	$\mathbf{10.6 \pm 0.1}$
CoinRun	9.4 ± 0.3	9.5 ± 0.2	5.4 ± 0.4	9.4 ± 0.2	$\mathbf{9.9 \pm 0.0}$	9.8 ± 0.0	9.8 ± 0.1	6.6 ± 0.4
Jumper	8.3 ± 0.2	8.5 ± 0.4	3.6 ± 0.5	8.2 ± 0.1	8.7 ± 0.1	8.6 ± 0.3	8.7 ± 0.2	$\mathbf{9.0 \pm 0.4}$
Chaser	4.1 ± 0.3	3.4 ± 0.9	6.3 ± 0.7	7.0 ± 0.6	$\mathbf{10.7 \pm 0.4}$	6.9 ± 1.2	7.5 ± 0.8	2.2 ± 0.1
Climber	6.9 ± 1.0	7.5 ± 0.8	6.2 ± 0.8	8.6 ± 0.6	10.2 ± 0.2	10.0 ± 0.3	$\mathbf{10.2 \pm 0.7}$	10.0 ± 0.3
Dodgeball	5.3 ± 2.3	9.1 ± 0.5	2.0 ± 1.1	7.3 ± 0.8	5.5 ± 0.5	5.2 ± 0.4	4.9 ± 0.3	$\mathbf{15.9 \pm 0.3}$
Heist	7.1 ± 0.5	4.4 ± 0.3	1.2 ± 0.4	6.2 ± 0.6	$\mathbf{7.4 \pm 0.4}$	5.2 ± 0.7	4.5 ± 0.3	7.2 ± 0.1
Leaper	5.5 ± 0.4	3.2 ± 1.2	6.4 ± 0.4	5.0 ± 0.9	$\mathbf{9.3 \pm 1.1}$	8.0 ± 1.1	8.3 ± 0.7	9.0 ± 0.3
Maze	$\mathbf{9.1 \pm 0.2}$	8.7 ± 0.7	4.1 ± 0.5	8.5 ± 0.3	9.0 ± 0.2	6.6 ± 0.4	6.4 ± 0.5	5.7 ± 0.9
Miner	11.7 ± 0.5	8.9 ± 0.9	9.7 ± 0.4	$\mathbf{12.0 \pm 0.3}$	11.3 ± 1.0	11.3 ± 0.9	11.5 ± 0.5	1.1 ± 0.2

Table 3: Procgen scores on train levels after training on 25M environment steps. The mean and standard deviation are computed using 5 runs with different seeds.

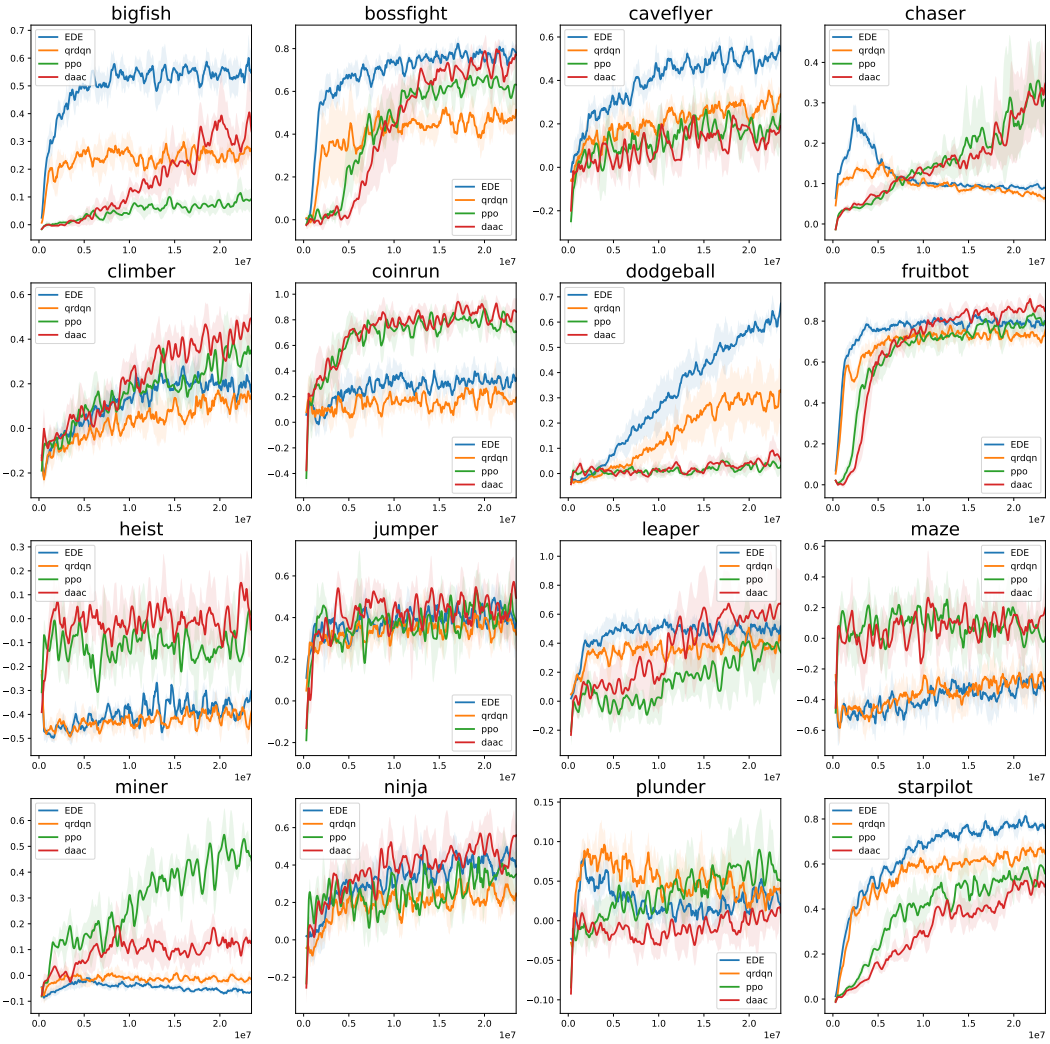


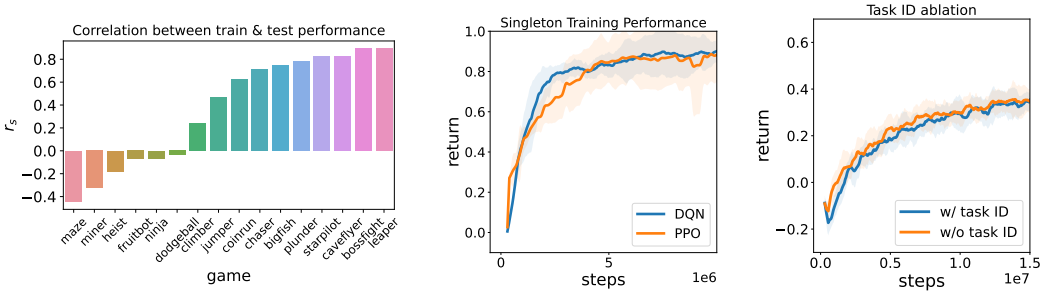
Figure 10: Min-max normalized test performance of a few representative methods on individual Procgen games. Mean and standard deviation are computed over 5 seeds. We can see that for games that value-based methods can make meaningful progress, EDE almost improve up QR-DQN. In many cases, EDE is significantly more sample efficient compared to the other methods. There are notably 4 games (`chaser`, `maze`, `heist`, `miner`) where value-based methods (including EDE) still perform much worse than policy optimization. These games all require generalization with long horizon planning which appear to remain challenging for value-based methods. Getting value-based methods to generalize on these games would be an interesting future direction.

E Additional Analysis

E.1 Other Sources of Poor Generalization

While our method improves train and test performance on two different CMDP benchmarks, it still does not fully close the generalization gap. In this section, we study other potential reasons for the poor generalization of RL algorithms, particularly value-based ones.

Optimization vs. Generalization. In supervised learning, deep neural networks have good generalization, meaning that they perform similarly well on both train and test data. However, in RL, the main bottleneck remains optimization rather than generalization. For example in Procgen, the state-of-the-art *training* performance is far below the theoretical optimum (Cobbe et al., 2019; Raileanu & Fergus, 2021), suggesting that current RL algorithms do not yet "overfit" the training distribution. To verify this hypothesis, we first conduct a meta-analysis of existing algorithms. We choose 7 representative algorithms (Appendix D) and measure whether better performance on training MDPs



(a) Meta Analysis (b) Singleton Experiments (c) DQN with Task ID

Figure 11: Investigations of other potential sources of poor generalization in RL.

translates into better performance on test MDPs. More specifically, we compute the Spearman's rank correlation, r_s, between the training and test performance of the algorithms, for each Procgen environment (Figure 11a). For many Procgen environments, training performance is strongly correlated with test performance, suggesting that further improvements in optimization could also result in better generalization.

In Figure 12a and Figure 12b, we show the scatter plot of the training performace and test performance of each algorithm on games with positive correlation and games with negative correlation (each color represents a different game). For games with positive correlation, we see that the correlation is almost linear with slope of 1 (*i.e.,* almost no generalization gap), which suggests that for these games, we can expect the good generalization if the training performance is good. On the games with negative correlation, it is less clear that improving training performance will necessarily lead to better generalization or more severe overfitting. Most of the games with negative correlation are either saturated in terms of performance or too challenging for current methods to make much progress. This suggests that despite their apparent similarity, there are actually two "categories" of games based on the algorithm's tendency to overfit. Note that this dichotomy likely depends on the algorithm; here, all algorithms are PPO-based and we hypothesize the picture would be different for value-based methods. It would be interesting to understand what structural properties of these CMDPs are responsible for this difference in overfitting, but this is out of scope for the current work.

Another interesting thing to notice is that most of the outliers in the two plots are PLR (Jiang et al., 2021) which is an *adversarial* method that actively tries to reduce training performance. This may explain why the trend does not hold for PLR as all other methods use a uniform distribution over the training environments.

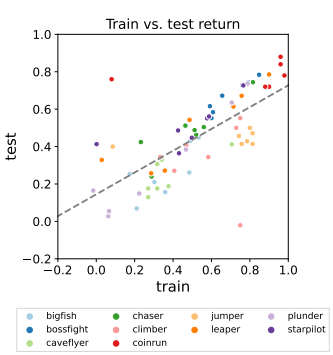
Sample Efficiency. One hypothesis for the poor performance of value-based methods relative to policy optimization ones in CMDPs is that the former is less sample efficient than the latter in each individual MDP $\mu \in \mathcal{M}_{\text{train}}$. If that is the case, the suboptimality can accumulate across all the training MDPs, resulting in a large gap between these types of methods when trained in all the environments in $\mathcal{M}_{\text{train}}$. To test this hypothesis, we run both DQN and PPO on a *single* environment from each of the Procgen games. As shown in Figure 11b, we see that there is not a significant gap between the average performances of DQN and PPO on single environments (across 3 seeds for 10 environments on which both methods reach scores above 0.5). In fact, DQN is usually slightly more sample-efficient. Thus, we can rule this out as a major factor behind the observed discrepancy.

Partial Observability. Another sensible hypothesis for why DQN underperforms PPO in CMDPs is the partial observability due to not knowing which environment the agent is interacting with at any given time (Ghosh et al., 2021). Policy optimization methods that use trajectory-wise Monte Carlo update are less susceptible to partial observability whereas value-based methods that use temporal difference updates can suffer more since they rely heavily on the Markovian assumption. To test this hypothesis, we provide the task ID to the Q-network in addition to the observation similar to Liu et al. (2021). The access to the task ID means the agent knows exactly which environment it is in (even though the environment itself may still be partially observable like Atari). In Figure 11b, we see that both methods do well on the singleton environments, so with task IDs, the algorithms should in theory be able to do as well as singleton environments even if there is no transfer between different MDPs because the model can learn them separately. In practice, we embed the discrete task IDs into $\mathbb{R}^{64}$ and add them as input to the final layers of the Q-network. Since there is no semantic

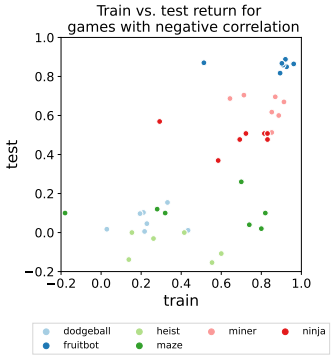
relationship between discrete task IDs, we do not expect this to improve generalization performance, but, surprisingly, we found that it *does not* improve training performance either (Figure 11c). This suggests that partial observability may not be the main problem in such environments as far as training is concerned. Note that this issue is related to but not the same as the aforementioned value-overfitting issue. Having task ID means that the agent *can* have an accurate point estimate for each MDP (as far as representation is concerned), but optimization remains challenging without proper exploration.

Value Overfitting. Most deep RL algorithms model value functions as point estimates of the expected return at each state. As discussed in Raileanu & Fergus (2021), this is problematic in CMDPs because the same state can have different values in different environments. Hence, the only way for the values to be accurate is to rely on spurious features which are irrelevant for finding the optimal action. For policy optimization algorithms, the policy can be protected from this type of overfitting by using separate networks to train the policy and value, as proposed in Raileanu & Fergus (2021). However, this approach cannot be applied to value-based methods since the policy is directly defined by the Q-function. An alternative solution is *distributional RL* (Bellemare et al., 2017) which learns a distribution (rather than a point estimate) over all possible Q-values for a given state. This models the *aleatoric uncertainty* resulting from the unobserved contexts in CMDPs (*i.e.,* not knowing which MDP the agent is interacting with), thus being able to account for one state having different potential values depending on the MDP. Our method uses distributional RL to mitigate the problem of value overfitting. As seen in Figure 5, while learning a value distribution leads to better results than learning only a point estimate (*i.e.,* QR-DQN $>$ DQN), the largest gains are due to using a more sophisticated exploration method (*i.e.,* QR-DQN + UCB + TNN $\gg$ DQN).

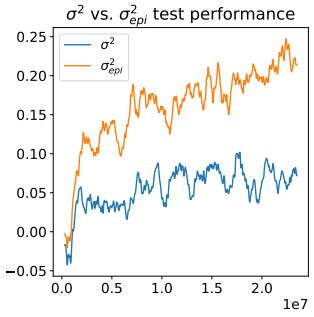
Our analysis indicates that sample efficiency and partial observability are not the main reasons behind the poor generalization of value-based methods. In contrast, value overfitting is indeed a problem, but it can be alleviated with distributional RL. Nevertheless, effective exploration proves to be a key factor in improving the train and test performance of value-based approaches in CMDPs. Our results also suggest that better optimization on the training environment can lead to better generalization to new environments, making it a promising research direction.



(a) Games with positive train-test correlation.



(b) Games with negative train-test correlation.



(c) Comparing different uncertainties (mean test min-max normalized scores).

Figure 12: Additional empirical results.

F UNCERTAINTY

Central to our method is the estimation of uncertainty in Q-values. We found that what kind of uncertainty to model and how we model it are crucial to the performance of EDE. The first important consideration is what kind of uncertainty to model. Specifically, we argue that only epistemic uncertainty is useful for exploration since they represent the reducible uncertainty. This is a longstanding view shared by many different sub-field of machine learning (Mockus, 1982; Settles, 2009). The aleatoric uncertainty on the other hand is intrinsic to the environment (*e.g.,* label noise) and cannot be reduced no matter how many samples we gather. In standard regression, even with ensembles, the estimated uncertainty (*i.e.,* variance) contains both epistemic and aleatoric uncertainty. In Atari, the main source of aleatoric uncertainty comes from the sticky action which has a relatively simple

structure. However, in CMDPs such as Procgen, the aleatoric uncertainty coming from having different environments is much more complicated and possibly multimodal, making it important to model them explicitly. Empirically, using aleatoric uncertainty to explore is detrimental to the test performance which may partially explain DQN+UCB's extremely poor performance (Figure 5), even compared to DQN+ε-greedy (*i.e.,* uncertainty estimation based on ensemble of non-distributional DQNs contains both types of uncertainties). To further test this hypothesis, we run QR-DQN+UCB with both aleatoric and epistemic uncertainty and a version with only epistemic uncertainty (Figure 12c). The results show that using only epistemic uncertainty performs significantly better than using both uncertainties in terms of both training and test performance.

Some additional important detail about uncertainty estimation with an ensemble are the number of ensembles and how the members of the ensemble are updated. These details have implications for both the computational complexity of inference and the quality of estimated uncertainty. For the main experiments, we use 5 ensemble members which, without specific optimization, results in about 2.5 times more time per algorithm step. The fol lowing are the time complexity of our algorithm on a Tesla V100:

- EDE (5 ensemble members): 0.3411 seconds per algorithm step
- EDE (3 ensemble members): 0.2331 seconds per algorithm step
- QR-DQN: 0.1356 seconds per algorithm step

In Figure 13b, we found that using 3 ensemble heads achieves comparable performance. On the other hand, using deep ensemble seems to be crucial for a good estimation of epistemic uncertainty as UA-DQN (Clements et al., 2019), which uses MAP sampling with 3 ensemble members, performs much worse than QR-DQN with 3 ensemble members trained with different minibatch and initialization (which are presumably more independent from each other). We believe further understanding this phenomenon would be an exciting future research direction.

G ARCHITECTURE AND HYPERPARAMETERS

G.1 PROCGEN

Architecture. We use a standard ResNet (He et al., 2016) feature extractor (f) from IMPALA (Espeholt et al., 2018), which was adopted by Ehrenberg et al. (2022) for value-based methods. The feature maps are flattened ($d = 2048$) and processed by M copies of 2-layer MLPs with the same architecture to output $M \times |\mathcal{A}| \times N$ values (for other models, this shape is changed accordingly for the appropriate output dimension). $|\mathcal{A}| = 15$ for all Procgen games. Each MLP conisists of two liner layers (both of dimension 512) separated by a ReLU activation; the first layer is $g^{(1)} : \mathbb{R}^d \to \mathbb{R}^{512}$ and the second layer is $g^{(2)} : \mathbb{R}^{512} \to \mathbb{R}^{|\mathcal{A}| \times N}$. The overall archetecture is

$$g \circ f = g^{(2)} \circ \texttt{ReLU} \circ g^{(1)} \circ \texttt{Flatten} \circ f.$$

The hyperparameters for the main method is shown in Table 4, and are directly adapted from Ehrenberg et al. (2022) other than those of new algorithmic components.

For hyperparameters specific to EDE, we search over the following values on the game `bigfish`:

- $\varphi : \{10, 20, 30, 50\}$
- $\lambda : \{0.5, 0.6, 0.7\}$
- $\alpha : \{6, 7, 8, 9\}$

and picked the best combination.

Points of comparisons. We try to keep the hyperaparameters the same as much as possible and only make necessary changes to keep the comparison fair:

- For base DQN and QR-DQN, we use ε-greedy exploration with decaying ε. Specifically, at every algorithmic step, the epsilon is computed with the following equation:

$$\varepsilon(t) = \min\left(0.1 + 0.9 \exp\left(-\frac{t - 2000}{8000}\right), 1\right)$$

Name	Value
number of training envs	200
procgen distribution mode	easy
minibatch size	512
number of parallel workers (K)	64
replay buffer size	10^6
discount factor (γ)	0.99
n-step return	3
frame stack	1
dueling network	no
target network update frequency	32000 (algo step)
initial random experience collection steps	$2000 \times K$ (env step)
total number of steps	25×10^6 (env step)
update per algo step	1
env step per algo step	K
Adam learning rate	2.5×10^{-4}
Adam epsilon	1.5×10^{-4}
prioritized experience replay	no
number of quantiles (N)	200
number of ensemble heads (M)	5
Huber loss κ	1
gradient clip norm	10
φ (UCB)	30
λ (TEE)	0.6
α (TEE)	7

Table 4: Hyperparameters for the main Procgen experiments. One algorithm step (algo step) can have multiple environment steps (env step) due to the distributed experience collection.

- For Bootstrapped DQN (Osband & Van Roy, 2015), we use bootstrapping to trained the model and uniformly sample a value head for each actor at the beginning of each episode that is used for the entire episode. We use 5 ensemble members just like the other methods.
- For UCB, we use the estimated epistemic uncertainty when combined with QR-DQN and use the standard variance when combined with DQN (Chen et al., 2017) and use $\varphi = 30$.
- For TEE without UCB, we use ε-greedy exploration *without* decaying ε for both QR-DQN and DQN, and EDE with the same λ and α.
- For ϵz-greedy, we use the same ε decay schedule in combination with $n = 10000$ and $\mu = 2.0$ which are the best performing hyperparameter values from Dabney et al. (2021). Each parallel worker has its own ϵz-greedy exploration process.
- For NoisyNet, we modify the base fully-connected layer with the noisy implementation (Fortunato et al., 2017) with $\sigma = 0.5$.

G.2 CRAFTER

For Crafter, we use the default architecture and hyperparameters that are used for the Rainbow experiments in Hafner (2022) with the following changes:

- Change the distributional RL component from C51 to QR-DQN with appropriate architecture change and use EDE for exploration
- Change the minibatch size from 32 to 64 to speed up training
- Remove prioritized experience replay

Once again, we emphasize that everything else stays the same, so the performance gain can be largely attributed to the proposed method. For EDE, since the default configuration from Hafner (2022) is single thread, we use Thompson sampling instead of UCB and do not use TEE.

For φ, we searched over $\{0.5, 1.0, 2.0\}$ and picked the best value.

Name	Value
minibatch size	64
number of parallel workers (K)	1
replay buffer size	10^6
discount factor (γ)	0.99
n-step return	3
frame stack	4
dueling network	no
target network update frequency	8000 (algo step)
initial random experience collection steps	20000 (env steps)
total number of steps	10^6 (env steps)
algo step per update	4
env step per algo step	1
Adam learning rate	6.25×10^{-5}
Adam epsilon	1.5×10^{-4}
prioritized experience replay	no
number of quantiles (N)	200
number of ensemble heads (M)	5
Huber loss κ	1
gradient clip norm	10
φ (Thompson sampling)	0.5

Table 5: Hyperparameters for the final Crafter experiments.

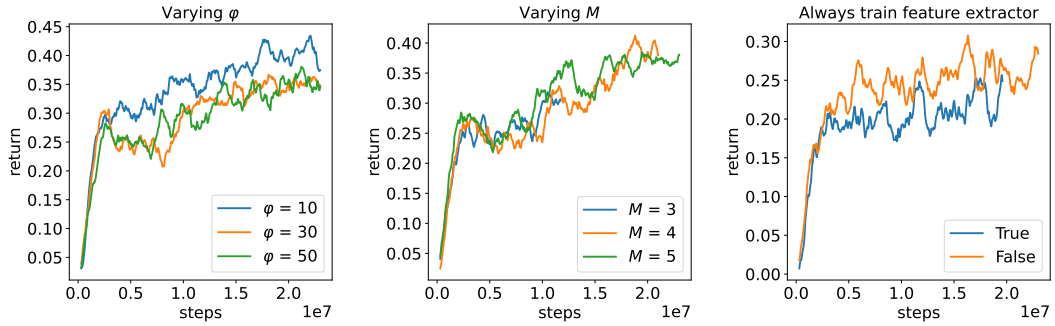
H SENSITIVITY STUDY

In this section, we study the sensitivity of the test performance (*i.e.,* aggregated min-max normalized scores) to various hyperparameters on a subset of games. First, *without TEE*, we study the sensitivity to different φ (UCB exploration coefficients), different M (number of ensemble members), and whether the feature extractor is trained with gradients from each ensemble member.

For φ and M, we run the algorithm on: `bossfight, ninja, plunder, starpilot, caveflyer, jumper, bigfish, leaper` for 1 seed and report the aggregated min-max normalized scores in Figure 13a and Figure 13b. We observe that the algorithm is slightly sensitive to the choice of φ, but not sensitive at all to M. This is encouraging since a large amount of computational overhead comes from the ensemble. Note that while smaller φ performs the best here, we use a larger value ($\varphi = 30$) in combination with TEE.

In Figure 13c, we show the effect of only training the feature extractor using the gradient from one member of the ensemble at every iteration. The results are computed on: `ninja, plunder, jumper, caveflyer, bigfish, leaper, climber` for 1 seed. We observe that always training the feature extractor leads to lower performance, corroborating our intuition that the feature extractor should be trained at the same speed as the individual ensemble members.

In Figure 14, we study the performance under different hyperparameter values of TEE. We use fixed $M = 5$ and $\varphi = 30$ and vary the values of either α or λ while holding the other one fixed. We observe no significant difference across these, suggesting that the algorithm is robust to the values of α and λ.



(a) φ ablation. (b) M ablation. (c) Feature extractor ablation.

Figure 13: Aggregated min-max normalized test scores for φ (for fixed $M = 3$, and training feature extractor for all value heads), M (for fixed $\varphi = 50$ and training feature extractor for all value heads), and whether feature extractor is trained with all value head (for fixed $\varphi = 50$ and $M = 3$).

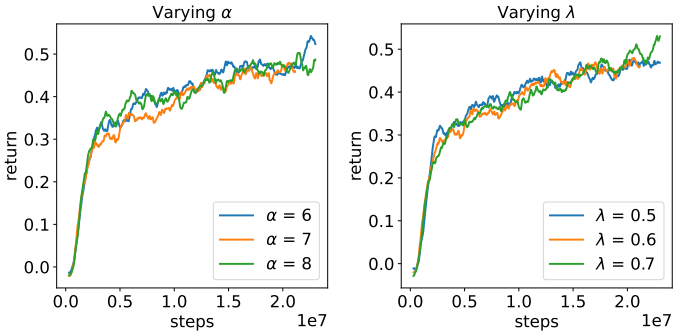


Figure 14: Aggregated min-max normalized test scores for λ (for fixed $\alpha = 7$) and α (for fixed $\lambda = 0.6$) on 4 games: `bossfight`, `climber`, `plunder`, `starpilot`.

I Case Study: Simplified Bigfish

To further illustrate the intuition that exploration on the training MDPs can help zero-shot generalization to other MDPs, we will use a slightly simplified example inspired by the game `bigfish` (Figure 15) from Procgen. As illustrated in Figure 16, in `bigfish`, the agent (green circle) needs to eat smaller fish (small red circle) and avoid bigger fish (large red circle). The images with solid borders are the observed frames and the images with dashed borders are the transitions.

If the agent collides with the small fish, it eats the small fish and gains +1 reward; if the agent collides with the big fish, it dies and the episode ends. The blue rectangle (top) represents the training environment and the red rectangle (bottom) represents the test environment. In the training MDP, the agent always starts in the state shown in the $T = 0$ of the greedy trajectory (top row). Using random exploration strategy (*e.g.,* ϵ-greedy), the agent should be able to quickly identify that going to the top will be able to achieve the $+1$ reward. However, there is an alternative trajectory (middle) where the agent can go down and right to eat the small fish on the right (orange border). This trajectory is longer (therefore harder to achieve via uniformly random exploration than the first one) and has lower discounted return.

From the perspective of solving the training MDP, the trajectory is suboptimal. If this trajectory is sufficiently hard to sample, the agent with a naive exploration will most likely keep repeating the greedy trajectory. On the other hand, once the agent has sampled the greedy trajectory sufficiently, uncertainty-aware exploration (*e.g.,* UCB or count-based exploration) will more likely sample these rarer trajectories since they have been visited less and thus have higher uncertainty. This has no

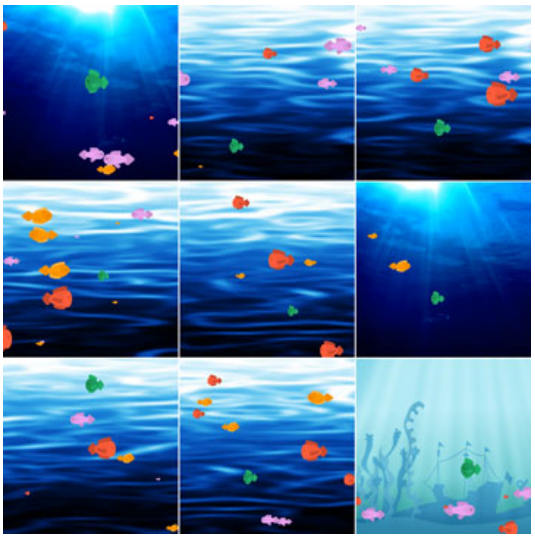


Figure 15: Example frames of `bigfish`. The goal for the agent (green fish) is to avoid fish bigger than itself and eat smaller fish. The spawning of enemy fish is different across different MDPs and the background can change.

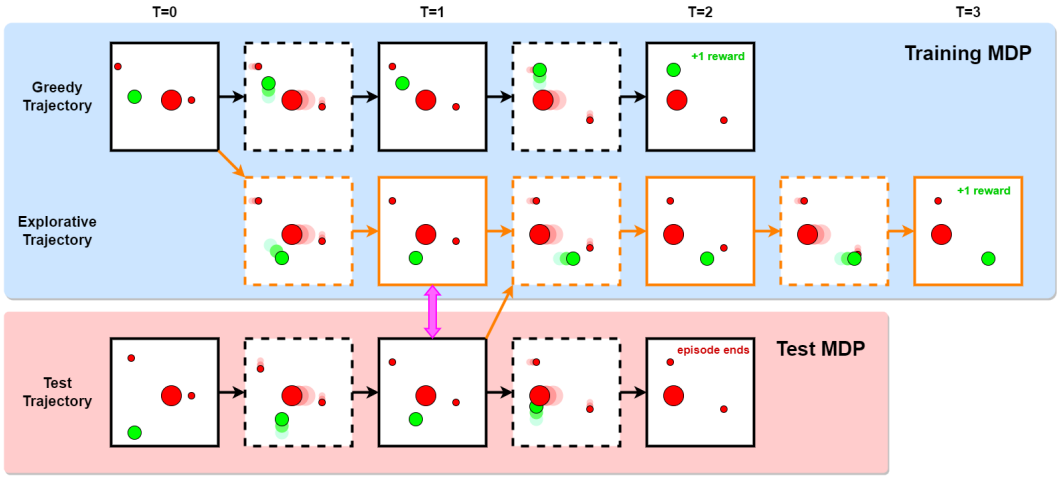


Figure 16: Simplified version of `bigfish`.

impact on the performance in the training MDP because the agent has learned the optimal policy regardless of which exploration strategy is used.[2]

However, on the test MDP, the initial state is shown in $T = 0$ of the test trajectory (bottom row). There is no guarantee that the agent knows how to behave correctly in this state because it has not seen it during training. One could hope that the neural network has learned a good representation so the agent knows what to do, but this does not necessarily have to be the case — the objective only cares about solving the training MDP and does not explicitly encourage learning representations that help generalization. Suppose that the agent keeps moving up due to randomness in the initialization of the weight, it will run into the big fish and die (bottom row, $T = 2$). Notice that even though the two environments are distinct, the test trajectory (bottom row) and the explorative trajectory (middle row) share the same state at $T = 1$ (pink arrow). Due to this structure, agents that have learned about the explorative trajectory during training time will be able to recognize that the better action from $T = 1$ is to move to the right which will avoid death and ultimately result in $+1$ reward.

Once again, we emphasize that this is a *simplified* example for illustrative purpose. In reality, the sequences of states do not need to be exactly the same and the neural network can learn to map

[2]Using function approximators with insufficient capacity may induce some effect on the optimal policy.

similar states to similar representations. In principle, the neural network can also learn to avoid the big fish but at $T = 0$ of the training MDP, the behavior "moving up" is indistinguishable from the behavior "avoiding the big fish". Good exploration during training can lead the agent to run into the big fish from all directions which is much closer to "avoid the big fish", the intuitive generalizable behavior. Clearly, not all CMDPs will have this structure which allows us to improve generalization via exploration, but it should not hurt. Like most problems in deep RL, it is difficult to prove this hypothesis analytically but empirically it is indeed the case that EDE has better or the same performance as QR-DQN in 11 out of 16 Procgen environments and perform approximately the same in the remaining environments (Figure 10).

This perspective allows us to tackle the problem with exploration that is in principle orthogonal to representation learning; however, since exploration naturally collects more diverse data, it may also be beneficial for representation learning as a side effect.