Self-Paced Deep Reinforcement Learning

Pascal Klink[1], Carlo D'Eramo[1], Jan Peters[1], Joni Pajarinen[1,2]
[1] Intelligent Autonomous Systems, Technische Universität Darmstadt, Germany
[2] Department of Electrical Engineering and Automation, Aalto University, Finland
Correspondence to: `pascal.klink@tu-darmstadt.de`

Abstract

Curriculum reinforcement learning (CRL) improves the learning speed and stability of an agent by exposing it to a tailored series of tasks throughout learning. Despite empirical successes, an open question in CRL is how to automatically generate a curriculum for a given reinforcement learning (RL) agent, avoiding manual design. In this paper, we propose an answer by interpreting the curriculum generation as an inference problem, where distributions over tasks are progressively learned to approach the target task. This approach leads to an automatic curriculum generation, whose *pace* is controlled by the agent, with solid theoretical motivation and easily integrated with deep RL algorithms. In the conducted experiments, the curricula generated with the proposed algorithm significantly improve learning performance across several environments and deep RL algorithms, matching or outperforming state-of-the-art existing CRL algorithms.

1 Introduction

Reinforcement learning (RL) [1] enables agents to learn sophisticated behaviors from interaction with an environment. Combinations of RL paradigms with powerful function approximators, commonly referred to as deep RL (DRL), have resulted in the acquisition of superhuman performance in various simulated domains [2, 3]. Despite these impressive results, DRL algorithms suffer from high sample complexity. Hence, a large body of research aims to reduce sample complexity by improving the explorative behavior of RL agents in a single task [4, 5, 6, 7].

Orthogonal to exploration methods, curriculum learning (CL) [8] for RL investigates the design of task sequences that maximally benefit the learning progress of an RL agent, by promoting the transfer of successful behavior between tasks in the sequence. To create a curriculum for a given problem, it is both necessary to define a set of tasks from which it can be generated and, based on that, specify *how* it is generated, i.e. how a task is selected given the current performance of the agent. This paper addresses the curriculum generation problem, assuming access to a set of parameterized tasks.

Recently, an increasing number of algorithms for curriculum generation have been proposed, empirically demonstrating that CL is an appropriate tool to improve the sample efficiency of DRL algorithms [9, 10]. However, these algorithms are based on heuristics and concepts that are, as of now, theoretically not well understood, preventing the establishment of rigorous improvements. In contrast, we propose to generate the curriculum based on a principled inference view on RL. Our approach generates the curriculum based on two quantities: The value function of the agent and the KL divergence to a target distribution of tasks. The resulting curriculum trades off task complexity (reflected in the value function) and the incorporation of desired tasks (reflected by the KL divergence). Our approach is conceptually similar to the self-paced learning (SPL) paradigm in supervised learning [11], which has only found application to RL in limited settings [12, 13].

Contribution We propose a new CRL algorithm, whose behavior is well explained as performing approximate inference on the common latent variable model (LVM) for RL [14, 15] (Section 4).

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

This enables principled improvements through the incorporation of advanced inference techniques. Combined with the well-known DRL algorithms TRPO, PPO and SAC [16, 17, 18], our method matches or surpasses the performance of state-of-the-art CRL methods in environments of different complexity and with sparse and dense rewards (Section 5).

2 Related Work

Simultaneously evolving the learning task with the learner has been investigated in a variety of fields ranging from behavioral psychology [19] to evolutionary robotics [20] and RL [21]. For supervised learning (SL), this principle was given the name *curriculum learning* [8]. This name has by now also been established in the RL community, where a variety of algorithms, aiming to generate curricula that maximally benefit the learner, have been proposed. Narvekar and Stone [22] showed that learning to create the *optimal* curriculum can be computationally harder than learning the entire task from scratch, motivating research on tractable approximations.

Keeping the agent's success rate within a certain range allowed to create curricula that drastically improve sample efficiency in tasks with binary reward functions or success indicators [23, 10, 24]. Many CRL methods [25, 26, 27, 28] have been proposed inspired by the idea of 'curiosity' or 'intrinsic motivation' [29, 30] – terms that refer to the way humans organize autonomous learning even in the absence of a task to be accomplished. Despite the empirical success, no theoretical foundation has been developed for the aforementioned methods, preventing principled improvements.

Another approach to curriculum generation has been explored under the name *self-paced learning* (SPL) for SL [11, 31, 32], proposing to generate a curriculum by optimizing the trade-off between exposing the learner to all available training samples and selecting samples in which it currently performs well. Despite its widespread application and empirical success in SL tasks, SPL has only been applied in a limited way to RL problems, restricting its use to the regression of the value function from an experience buffer [13] or to a strictly episodic RL setting [12]. Our method connects to this line of research, formulating the curriculum generation as a trade-off optimization of similar fashion. While the work by Ren et al. [13] is orthogonal to ours, we identify the result of Klink et al. [12] as a special case of our inference view. Besides allowing the combination of SPL and modern DRL algorithms to solve more complex tasks, the inference view presents a unified theory of using the self-paced learning paradigm for RL.

As we interpret RL from an inference perspective over the course of this paper, we wish to briefly point to several works employing this perspective [33, 34, 35, 15, 14]. Taking an inference perspective is beneficial when dealing with inverse problems or problems that require tractable approximations [36, 37]. For RL, it motivates regularization techniques such as the concept of maximum- or relative entropy [38, 18, 39] and stimulates the development of new, and interpretation of, existing algorithms from a common view [40, 41]. Due to a common language, algorithmic improvements on approximate inference [42, 43, 44] can be shared across domains.

3 Preliminaries

We formulate our approach in the domain of reinforcement learning (RL) for contextual Markov decision processes (CMDPs) [45, 46]. A CMDP is a tuple $<\mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{M}>$, where $\mathcal{M}(\boldsymbol{c})$ is a function that maps a context $\boldsymbol{c} \in \mathcal{C}$ to a Markov decision process (MDP) $\mathcal{M}(\boldsymbol{c}) {=} <\mathcal{S}, \mathcal{A}, p_{\boldsymbol{c}}, r_{\boldsymbol{c}}, p_{0,\boldsymbol{c}}>$. An MDP is an abstract environment with states $\boldsymbol{s} \in \mathcal{S}$, actions $\boldsymbol{a} \in \mathcal{A}$, transition probabilities $p_{\boldsymbol{c}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$, reward function $r_{\boldsymbol{c}} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ and initial state distribution $p_{0,\boldsymbol{c}}(\boldsymbol{s})$. Typically $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{C}$ are discrete spaces or subsets of $\mathbb{R}^n$. We can think of a CMDP as a parametric family of MDPs, which share the same state-action space. Such a parametric family of MDPs allows to share policies and representations between them [47], both being especially useful for CRL. RL for CMDPs encompasses approaches that aim to find a policy $\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{c})$ which maximizes the expected return over trajectories $\tau = \{(\boldsymbol{s}_t, \boldsymbol{a}_t)|t \geq 0\}$

$$J(\mu, \pi) {=} E_{\mu(\boldsymbol{c}), p_\pi(\tau|\boldsymbol{c})}\left[\sum_{t\geq 0} \gamma^t r_{\boldsymbol{c}}(\boldsymbol{s}_t, \boldsymbol{a}_t)\right],\ p_\pi(\tau|\boldsymbol{c}) {=} p_{0,\boldsymbol{c}}(\boldsymbol{s}_0) \prod_{t \geq 0} p_{\boldsymbol{c}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)\pi(\boldsymbol{a}_t|\boldsymbol{s}_t, \boldsymbol{c}), \quad (1)$$

with discount factor $\gamma \in [0, 1)$ and a probability distribution over contexts $\mu(\boldsymbol{c})$, encoding which contexts the agent is expected to encounter. We will often use the term policy and agent interchangeably, as the policy represents the behavior of a (possibly virtual) agent. RL algorithms parametrize the policy π with parameters $\boldsymbol{\omega} \in \mathbb{R}^n$. We will refer to this parametric policy as $\pi_{\boldsymbol{\omega}}$, sometimes replacing $\pi_{\boldsymbol{\omega}}$ by $\boldsymbol{\omega}$ in function arguments or subscripts, e.g. writing $J(\mu, \boldsymbol{\omega})$ or $p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c})$. The so-called value function encodes the expected long-term reward when following a policy $\pi_{\boldsymbol{\omega}}$ starting in state $\boldsymbol{s}$

$$V_{\boldsymbol{\omega}}(\boldsymbol{s},\boldsymbol{c}) = E_{p_{\boldsymbol{\omega}}(\tau|\boldsymbol{s},\boldsymbol{c})}\left[\sum_{t\geq 0}\gamma^t r_{\boldsymbol{c}}(\boldsymbol{s}_t,\boldsymbol{a}_t)\right],\ p_{\boldsymbol{\omega}}(\tau|\boldsymbol{s},\boldsymbol{c}) = \delta^{\boldsymbol{s}}_{\boldsymbol{s}_0}\prod_{t\geq 0} p_{\boldsymbol{c}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)\pi_{\boldsymbol{\omega}}(\boldsymbol{a}_t|\boldsymbol{s}_t,\boldsymbol{c}), \quad (2)$$

where $\delta^{\boldsymbol{s}}_{\boldsymbol{s}_0}$ is the delta-distribution. The above value function for CMDPs has been introduced as a general or universal value function [48, 47]. We will, however, just refer to it as a value function, since a CMDP can be expressed as an MDP with extended state space. We see that the value function relates to Eq. 1 via $J(\mu, \boldsymbol{\omega}) = E_{\mu(\boldsymbol{c}),p_{0,\boldsymbol{c}}(\boldsymbol{s}_0)}\left[V_{\boldsymbol{\omega}}(\boldsymbol{s}_0, \boldsymbol{c})\right]$.

4 Self-Paced Deep Reinforcement Learning

Having established the necessary notation, we now introduce a curriculum to the contextual RL objective (Eq. 1) by allowing the agent to choose a distribution of tasks $p_{\boldsymbol{\nu}}(\boldsymbol{c})$, parameterized by $\boldsymbol{\nu} \in \mathbb{R}^m$, to train on. Put differently, we allow the RL agent to maximize $J(p_{\boldsymbol{\nu}}, \pi)$ under a chosen $p_{\boldsymbol{\nu}}$, only ultimately requiring it to match the "desired" task distribution $\mu(\boldsymbol{c})$ to ensure that the policy is indeed a local maximizer of $J(\mu, \pi)$. We achieve this by reformulating the RL objective as

$$\max_{\boldsymbol{\nu},\boldsymbol{\omega}} J(\boldsymbol{\nu}, \boldsymbol{\omega}) - \alpha D_{\text{KL}}\left(p_{\boldsymbol{\nu}}(\boldsymbol{c})\|\mu(\boldsymbol{c})\right), \quad \alpha \geq 0. \quad (3)$$

In the above objective, $D_{\text{KL}}\left(\cdot\|\cdot\right)$ is the KL divergence between two probability distributions. The parameter α controls the aforementioned trade-off between freely choosing $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ and matching $\mu(\boldsymbol{c})$. When only optimizing Objective (3) w.r.t. $\boldsymbol{\omega}$ for a given $\boldsymbol{\nu}$, we simply optimize the contextual RL objective (Eq. 1) over the context distribution $p_{\boldsymbol{\nu}}(\boldsymbol{c})$. On the contrary, if Objective (3) is only optimized w.r.t. $\boldsymbol{\nu}$ for a given policy $\pi_{\boldsymbol{\omega}}$, then α controls the trade-off between incorporating tasks in which the policy obtains high reward and matching $\mu(\boldsymbol{c})$. So if we optimize Objective (3) in a block-coordinate ascent manner, we may use standard RL algorithms to train the policy under fixed $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ and then adjust $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ according to the obtained policy. If we keep increasing α during this procedure, $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ will ultimately match $\mu(\boldsymbol{c})$ due to the KL divergence penalty and we train on the true objective. The benefit of such an interpolation between task distributions under which the agent initially performs well and $\mu(\boldsymbol{c})$ is that the agent may be able to adapt well-performing behavior as the environments gradually transform. This can, in turn, avoid learning poor behavior and increase learning speed. The outlined idea resembles the paradigm of self-paced learning for supervised learning [11], where a regression- or classification model as well as its training set are alternatingly optimized. The training set for a given model is chosen by trading-off favoring samples under which the model has low prediction error and incorporating all samples in the dataset. Indeed, the idea of generating curricula for RL using the self-paced learning paradigm has previously been investigated by Klink et al. [12]. However, they investigate the curriculum generation only in the episodic RL setting and jointly update the policy and context distribution. This ties the curriculum generation to a specific (episodic) RL algorithm, that, as we will see in the experiments, is not suited for high-dimensional policy parameterizations. Our formulation is not limited to such a specific setting, allowing to use the resulting algorithm for curriculum generation with any RL algorithm. Indeed, we will now relate the maximization of Objective (3) w.r.t. $\boldsymbol{\nu}$ to an inference perspective, showing that our formulation explains the results obtained by Klink et al. [12].

Interpretation as Inference Objective (3) can be motivated by taking an inference perspective on RL [14]. In this inference perspective, we introduce an 'optimality' event $\mathcal{O}$, whose probability of occurring is defined via a monotonic transformation $f : \mathbb{R} \mapsto \mathbb{R}_{\geq 0}$ of the cumulative reward $R(\tau, \boldsymbol{c}) = \sum_{t\geq 0} r_{\boldsymbol{c}}(\boldsymbol{s}_t, \boldsymbol{a}_t)$, yielding the following latent variable model (LVM)

$$p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O}) = \int p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O}, \tau, \boldsymbol{c})d\tau d\boldsymbol{c} \propto \int f(R(\tau, \boldsymbol{c}))p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c})p_{\boldsymbol{\nu}}(\boldsymbol{c})d\tau d\boldsymbol{c}. \quad (4)$$

Under appropriate choice of $f(\cdot)$ and minor modification of the transition dynamics to account for the discounting factor γ [15, 14], maximizing LVM (4) w.r.t. $\boldsymbol{\omega}$ is equal to the maximization of $J(p_{\boldsymbol{\nu}}, \pi_{\boldsymbol{\omega}})$ w.r.t. $\pi_{\boldsymbol{\omega}}$. This setting is well explored and allowed to identify various RL algorithms as approximate applications of the expectation maximization (EM) algorithm to LVM (4) to maximize $p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O})$ w.r.t $\boldsymbol{\omega}$ [40]. Our idea of maximizing Objective (3) in a block-coordinate ascent manner is readily supported by the EM algorithm, since its steps can be executed alternatingly w.r.t. $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$. Consequently, we now investigate the case when maximizing $p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O})$ w.r.t. $\boldsymbol{\nu}$, showing that known regularization techniques for approximate inference motivate Objective (3). For brevity, we only state the main results here and refer to Appendix A for detailed proofs and explanations of EM.

When maximizing $p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O})$ w.r.t. $\boldsymbol{\nu}$ using EM, we introduce a variational distribution $q(\boldsymbol{c})$ and alternate between the so called E-Step, in which $q(\boldsymbol{c})$ is found by minimizing $D_{\text{KL}}\left(q(\boldsymbol{c})\|p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})\right)$, and the M-Step, in which $\boldsymbol{\nu}$ is found by minimizing the KL divergence to the previously obtained variational distribution $D_{\text{KL}}\left(q(\boldsymbol{c})\|p_{\boldsymbol{\nu}}(\boldsymbol{c})\right)$. Typically $q(\boldsymbol{c})$ is not restricted to a parametric form and hence matches $p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})$ after the E-Step. We now state our main theoretical insight, showing exactly what approximations and modifications to the regular EM algorithm are required to retrieve Objective 3.

Theorem 1. *Choosing $f(\cdot){=}\exp(\cdot)$, maximizing Objective (3) minus a KL divergence term $D_{KL}\left(p_{\boldsymbol{\nu}}(\boldsymbol{c})\|p_{\tilde{\boldsymbol{\nu}}}(\boldsymbol{c})\right)$ is equal to executing E- and M-Step while restricting $q(\boldsymbol{c})$ to be of the same parametric form as $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ and introducing a regularized E-Step $D_{KL}\left(q(\boldsymbol{c})\middle\|\frac{1}{Z}p_{\tilde{\boldsymbol{\nu}},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})^{\frac{1}{1+\alpha}}\mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}}\right)$.*

Theorem 1 is interesting for many reasons. Firstly, the extra term in Objective (3) can be identified as a regularization term, which penalizes a large deviation of $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ from $p_{\tilde{\boldsymbol{\nu}}}(\boldsymbol{c})$. In the algorithm we propose, we replace this penalty term by a constraint on the KL divergence between successive context distributions, granting explicit control over their dissimilarity. This is beneficial when estimating expectations in Objective (3) by a finite amount of samples. Next, restricting the variational distribution to a parametric form is a known concept in RL. Abdolmaleki et al. [40] have shown that it yields an explanation for the well-known on-policy algorithms TRPO and PPO [16, 17]. Finally, the regularized E-Step fits $q(\boldsymbol{c})$ to a distribution that is referred to as a *tempered* posterior. Tempering, or *deterministic annealing*, is used in variational inference to improve the approximation of posterior distributions by gradually moving from the prior (in our case $\mu(\boldsymbol{c})$) to the true posterior (here $p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})$) [44, 49, 50], which in above equation corresponds to gradually decreasing α to zero. We, however, increasingly enforce $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ to match $\mu(\boldsymbol{c})$ by gradually increasing α. To understand this "inverse" behavior, we need to remember that the maximization w.r.t. $\boldsymbol{\nu}$ solely aims to generate context distributions $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ that facilitate the maximization of $J(\mu,\pi_{\boldsymbol{\omega}})$ w.r.t. $\boldsymbol{\omega}$. This means to initially encode contexts in which the event $\mathcal{O}$ is most likely, i.e. $p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})$ is highest, and only gradually match $\mu(\boldsymbol{c})$. To conclude this theoretical section, we note that the update rule proposed by Klink et al. [12] can be recovered from our formulation.

Theorem 2. *Choosing $f(\cdot){=}\exp(\cdot/\eta)$, the (unrestricted) variational distribution after the regularized E-Step is given by $q(\boldsymbol{c}){\propto}p_{\boldsymbol{\nu}}(\boldsymbol{c})\exp\left(\frac{V_{\boldsymbol{\omega}}(\boldsymbol{c})+\eta\alpha(\log(\mu(\boldsymbol{c}))-\log(p_{\boldsymbol{\nu}}(\boldsymbol{c})))}{\eta+\eta\alpha}\right)$, where $V_{\boldsymbol{\omega}}(\boldsymbol{c})$ is the 'episodic value function' as defined in [34].*

The variational distribution in Theorem 2 resembles the results in [12] with the only difference that α is scaled by η. Hence, for a given schedule of α, we simply need to scale every value in this schedule by $1/\eta$ to match the results from Klink et al. [12].

Algorithmic Realization As previously mentioned, we maximize Objective (3) in a block-coordinate ascent manner, i.e. use standard RL algorithms to optimize $J(p_{\boldsymbol{\nu}_i},\pi_{\boldsymbol{\omega}})$ w.r.t. $\pi_{\boldsymbol{\omega}}$ under the current context distribution $p_{\boldsymbol{\nu}_i}$. Consequently, we only need to develop ways to optimize Objective (3) w.r.t. $p_{\boldsymbol{\nu}}$ for a given policy $\pi_{\boldsymbol{\omega}_i}$. We can run any RL algorithm to generate a set of trajectories $\mathcal{D}_i{=}\left\{(\boldsymbol{c}^k,\tau^k)\middle|k\in[1,K],\boldsymbol{c}_k\sim p_{\boldsymbol{\nu}_i}(\boldsymbol{c}),\tau_k\sim\pi_{\boldsymbol{\omega}_i}(\tau|\boldsymbol{c}_k)\right\}$ alongside an improved policy $\pi_{\boldsymbol{\omega}_{i+1}}$. Furthermore, most state-of-the-art RL algorithms fit a value function $V_{\boldsymbol{\omega}_{i+1}}(\boldsymbol{s},\boldsymbol{c})$ while generating the policy $\pi_{\boldsymbol{\omega}_{i+1}}$. Even if the employed RL algorithm does not generate an estimate of $V_{\boldsymbol{\omega}_{i+1}}(\boldsymbol{s},\boldsymbol{c})$, it is easy to compute one using standard techniques. We can exploit the connection between value function and RL objective $J(p,\boldsymbol{\omega}_{i+1})=E_{p(\boldsymbol{c}),p_{0,\boldsymbol{c}}(\boldsymbol{s}_0)}\left[V_{\boldsymbol{\omega}_{i+1}}(\boldsymbol{s}_0,\boldsymbol{c})\right]$ to optimize

$$\max_{\boldsymbol{\nu}_{i+1}}\frac{1}{K}\sum_{k=1}^{K}\frac{p_{\boldsymbol{\nu}_{i+1}}\left(\boldsymbol{c}^k\right)}{p_{\boldsymbol{\nu}_i}\left(\boldsymbol{c}^k\right)}V_{\boldsymbol{\omega}_{i+1}}\left(\boldsymbol{s}_0^k,\boldsymbol{c}^k\right)-\alpha_i D_{\text{KL}}\left(p_{\boldsymbol{\nu}_{i+1}}(\boldsymbol{c})\middle\|\mu(\boldsymbol{c})\right)\quad\text{s.t. }D_{\text{KL}}\left(p_{\boldsymbol{\nu}_{i+1}}(\boldsymbol{c})\middle\|p_{\boldsymbol{\nu}_i}(\boldsymbol{c})\right){\leq}\epsilon. \tag{5}$$

Algorithm 1 Self-Paced Deep Reinforcement Learning

```
Input: Initial context distribution- and policy parameters $\boldsymbol{\nu}_0$ and $\boldsymbol{\omega}_0$, Target context distribution $\mu(\boldsymbol{c})$, KL penalty proportion $\zeta$, offset $N_\alpha$, number of iterations $N$, Rollouts per policy update $K$
for $i = 1$ to $N$ do
    Agent Improvement:
    Sample contexts: $\boldsymbol{c}^k \sim p_{\boldsymbol{\nu}_i}(\boldsymbol{c}),\ k = 1, \dots, K$
    Rollout trajectories: $\tau^k \sim \pi_{\boldsymbol{\omega}_i}(\tau|\boldsymbol{c}_k),\ k = 1, \dots, K$
    Obtain $\pi_{\boldsymbol{\omega}_{i+1}}$ from RL algorithm of choice using $\mathcal{D}_i = \left\{(\boldsymbol{c}^k, \tau^k)\middle|k = 1, \dots, K\right\}$
    Estimate $V_{\boldsymbol{\omega}_{i+1}}(\boldsymbol{s}_0^k, \boldsymbol{c}^k)$ for contexts $\boldsymbol{c}^k$ (using the employed RL algorithm, if possible)
    Context Distribution Update:
    Obtain $p_{\boldsymbol{\nu}_{i+1}}$ optimizing (Eq. 5), using $\alpha_i = 0,$ if $i \leq N_\alpha,$ else $\mathcal{B}(\boldsymbol{\nu}_i, \mathcal{D}_i)$ (Eq. 6)
end for
```

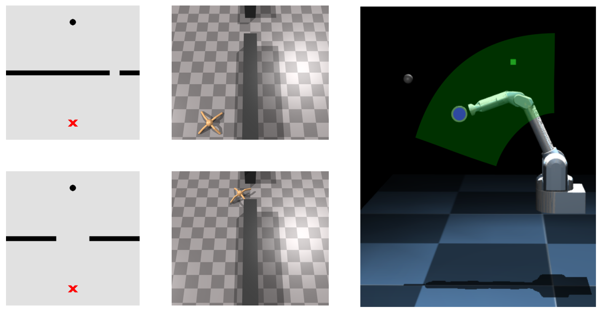
instead of Objective (3) to obtain $\boldsymbol{\nu}_{i+1}$. The first term in Objective (5) is an importance-weighted approximation of $J(\boldsymbol{\nu}_{i+1}, \boldsymbol{\omega}_{i+1})$. Motivated by Theorem 1, the KL divergence constraint between subsequent context distributions $p_{\boldsymbol{\nu}_i}(\boldsymbol{c})$ and $p_{\boldsymbol{\nu}_{i+1}}(\boldsymbol{c})$ avoids large jumps in the context distribution. Above objective can be solved using any constrained optimization algorithm. In our implementation, we use the trust-region algorithm implemented in the SciPy library [51]. In each iteration, the parameter α_i is chosen such that the KL divergence penalty w.r.t. the current context distribution is in constant proportion ζ to the average reward obtained during the last iteration of policy optimization

$$\alpha_i = \mathcal{B}(\boldsymbol{\nu}_i, \mathcal{D}_i) = \zeta \frac{\frac{1}{K}\sum_{k=1}^{K} R\left(\tau^k, \boldsymbol{c}^k\right)}{D_{\text{KL}}\left(p_{\boldsymbol{\nu}_i}(\boldsymbol{c}) \| \mu(\boldsymbol{c})\right)}, \quad R\left(\tau^k, \boldsymbol{c}^k\right) = \sum_{t\geq 0}\gamma^t r_{\boldsymbol{c}^k}\left(\boldsymbol{s}_t^k, \boldsymbol{a}_t^k\right), \tag{6}$$

as proposed by Klink et al. [12]. We further adopt their strategy of setting α to zero for the first N_α iterations. This allows to taylor the context distribution to the learner in early iterations, if the initial context distribution is uninformative, i.e. covers large parts of the context space. Note that this is a naive choice, that nonetheless worked sufficiently well in our experiments. At this point, the connection to tempered inference allows for principled future improvements by using more advanced methods to choose α [44]. For the experiments, we restrict $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ to be Gaussian. Consequently, Objective (5) is optimized w.r.t. the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ of the context distribution. Again, the inference view readily motivates future improvements by using advanced sampling techniques [43, 42]. These techniques allow to directly sample from the variational distribution $q(\boldsymbol{c})$ in Theorem 1, bypassing the need to fit a parametric distribution and allowing to represent multi-modal distributions. The outlined method is summarized in Algorithm 1.

5 Experiments

The aim of this section is to investigate the performance and versatility of the proposed curriculum reinforcement learning algorithm (SPDL). To accomplish this, we evaluate SPDL in three different environments with different DRL algorithms to test the proposition that the learning scheme benefits the performance of various RL algorithms. We evaluate the performance using TRPO [16], PPO [17] and SAC [18]. For all DRL algorithms, we use the implementations provided in the `Stable Baselines` library [52]. [1]



(a) Point-Mass (b) Ant (c) Ball-Catching

Figure 1: Environments used for experimental evaluation. For the point mass environment (a), the upper plot shows the target task. The shaded areas in picture (c) visualize the target distribution of ball positions (green) as well as the ball positions for which the initial policy succeeds (blue).

The first two environments aim at investigating the benefit of SPDL when the purpose of the generated curriculum is solely to facilitate the learning

[1] Code for running the experiments can be found at `https://github.com/psclklnk/spdl`

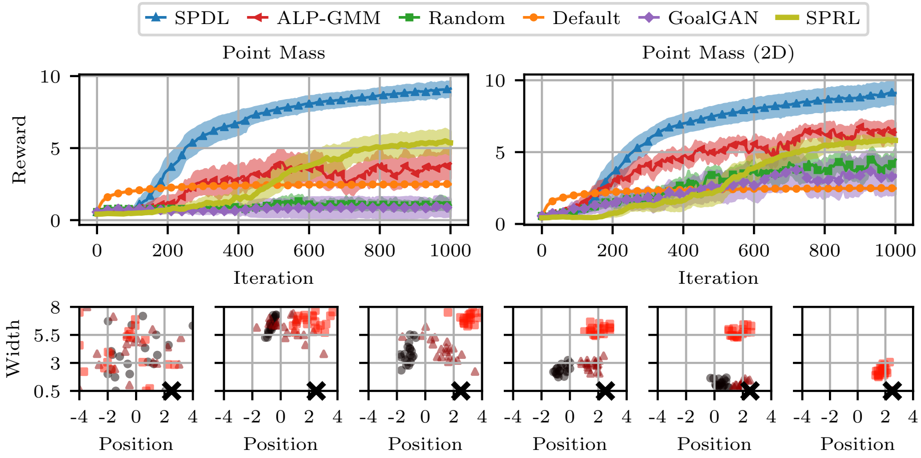


Figure 2: Reward of different curricula in the Point Mass (2D and 3D) environment for TRPO. Mean (thick line) and two times standard error (shaded area) is computed from 20 algorithm runs. The lower plots show samples from the context distributions $p(\boldsymbol{c})$ in the Point-Mass 2D environment at iterations 0, 50, 80, 100, 120 and 200 (from left to right). Different colors and shapes of samples indicate different algorithm runs. The black cross marks the mean of the target distribution $\mu(\boldsymbol{c})$.

of a hard target task, which the agent is not able to solve without a curriculum. For this purpose, we create two environments that are conceptually similar to the point mass experiment considered by SPRL [12]. The first one is a copy of the original experiment, but with an additional parameter to the context space, as we will detail in the corresponding section. The second environment extends the original experiment by replacing the point mass with a torque-controlled quadruped 'ant'. This increases the complexity of the underlying control problem, requiring the capacity of deep neural network function approximators used in DRL algorithms. The final environment is a robotic ball-catching environment. This environment constitutes a shift in curriculum paradigm as well as reward function. Instead of guiding learning towards a specific target task, this third environment requires to learn a ball-catching policy over a wide range of initial states (ball position and velocity). The reward function is sparse compared to the dense ones employed in the first two environments.

To judge the performance of SPDL, we compare the obtained results to state-of-the-art CRL algorithms ALP-GMM [27], which is based on the concept of Intrinsic Motivation, GoalGAN [23], which relies on the notion of a success indicator to define a curriculum, and SPRL [12], the episodic counterpart of our algorithm. Furthermore, we compare to curricula consisting of tasks uniformly sampled from the context space (referred to as 'Random' in the plots) and learning without a curriculum (referred to as 'Default'). Additional details on the experiments as well as qualitative evaluations of them can be found in Appendix B.

5.1 Point Mass Environment

In this environment, the agent controls a point mass that needs to be navigated through a gate of given size and position to reach a desired target in a two-dimensional world. If the point mass crashes into the wall, the experiment is stopped. The agent moves the point mass by applying forces and the reward decays in a squared exponential manner with increasing distance to the goal. In our version of the experiment, the contextual variable $\boldsymbol{c} \in \mathbb{R}^3$ changes the width and position of the gate as well as the dynamic friction coefficient of the ground on which the point mass slides. The target context distribution $\mu(\boldsymbol{c})$ is a narrow Gaussian with negligible noise that encodes a small gate at a specific position and a dynamic friction coefficient of 0. Figure 1 shows two different instances of the environment, one of them being the target task.

Figure 2 shows the results of two different experiments in this environment, one where the curriculum is generated over the full context space and one in which the friction parameter is fixed to its target value of 0. As Figure 2 and Table 1 indicate, SPDL significantly increases the asymptotic reward on the target task compared to all other methods. Furthermore, we see that SPRL, which we applied by defining the episodic RL policy $p(\boldsymbol{\omega}|\boldsymbol{c})$ to choose the weights $\boldsymbol{\omega}$ of the policy network for a

Table 1: Average final reward and standard error of different curricula and RL algorithms in the two Point Mass environments with three (P3D) and two (P2D) context dimensions as well as the Ball-Catching environment (BC). The data is computed from 20 algorithm runs. Significantly better results according to a t-test with $p < 1\%$ are highlighted in bold. The asterisks mark runs of SPDL/GoalGAN with an initialized context distribution and runs of Default learning without policy initialization.

	PPO (P3D)	SAC (P3D)	PPO (P2D)	SAC (P2D)	TRPO (BC)	PPO (BC)
ALP-GMM	2.34 ± 0.2	0.96 ± 0.3	5.24 ± 0.4	1.15 ± 0.4	39.8 ± 1.1	46.5 ± 0.7
GoalGAN	0.50 ± 0.0	1.08 ± 0.4	1.39 ± 0.5	0.72 ± 0.2	42.5 ± 1.6	42.6 ± 2.7
GoalGAN*	-	-	-	-	45.8 ± 1.0	45.9 ± 1.0
SPDL	$\mathbf{9.35 \pm 0.1}$	$\mathbf{4.43 \pm 0.7}$	$\mathbf{9.02 \pm 0.4}$	$\mathbf{4.69 \pm 0.7}$	47.0 ± 2.0	$\mathbf{53.9 \pm 0.4}$
SPDL*	-	-	-	-	43.3 ± 2.0	49.3 ± 1.4
Random	0.53 ± 0.0	0.60 ± 0.1	1.34 ± 0.3	0.93 ± 0.3	-	-
Default	2.46 ± 0.0	2.26 ± 0.0	2.47 ± 0.0	2.23 ± 0.0	21.0 ± 0.3	22.1 ± 0.3
Default*	-	-	-	-	21.2 ± 0.3	23.0 ± 0.7

given context $\boldsymbol{c}$, also leads to a good performance. Increasing the dimension of the context space has a stronger negative impact on the performance of the other CL algorithms than on both SPDL and SPRL, where it only negligibly decreases the performance. We suspect that this effect arises because both ALP-GMM and GoalGAN have no notion of a target distribution. Consequently, for a context distribution $\mu(\boldsymbol{c})$ with negligible variance, a higher context dimension decreases the average proximity of sampled tasks to the target one. By having a notion of a target distribution, SPDL ultimately samples contexts that are close to the desired ones, regardless of the dimension. The context distributions $p(\boldsymbol{c})$ visualized in Figure 2 show that the agent focuses on wide gates in a variety of positions in early iterations. Subsequently, the size of the gate is decreased and the position of the gate is shifted to match the target one. This process is carried out at different pace and in different ways, sometimes preferring to first shrink the width of the gate before moving its position while sometimes doing both simultaneously.

5.2 Ant Environment

We replace the point mass in the previous environment with a four-legged ant similar to the one in the OpenAI Gym simulation environment [53]. [2] The goal is to reach the other side of a wall by passing through a gate, whose width and position is determined by the contextual variable $\boldsymbol{c} \in \mathbb{R}^2$ (Figure 1).

Opposed to the previous environment, an application of SPRL is not straightforward in this environment, since the episodic policy needs to choose weights for a policy network with 6464 parameters. In such high-dimensional spaces, fitting the new episodic policy (i.e. a 6464-dimensional Gaussian) to the generated samples requires significantly more computation time than an update of a step-based policy, taking up to 25 minutes per update on our hardware. Furthermore, this step is prone to numerical instabilities due to the large covariance matrix that needs to be estimated. This observation stresses the benefit of our CRL approach, as it unifies the curriculum generation for episodic and step-based RL algorithms, allowing to choose the most beneficial one for the task at hand.

In this environment, we were only able to evaluate the CL algorithms using PPO. This is because the implementations of TRPO and SAC in the `Stable-Baselines` library do not allow to make use of the parallelization capabilities of the Isaac Gym simulator, leading to prohibitive running times (details in Appendix B).

Looking at Figure 3, we see that SPDL allows the learning agent to escape the local optimum which results from the agent not finding the gate to pass through. ALP-GMM and a random curriculum do not improve the reward over directly learning on the target task. However, as we show in Appendix B, both ALP-GMM and a random curriculum improve the qualitative performance, as they sometimes allow to move the ant through the gate. Nonetheless, this behavior is unreliable and inefficient, causing the action penalties in combination with the discount factor to prevent this better behavior from being reflected in the reward.

[2] We use the Nvidia Isaac Gym simulator [54] for this experiment.

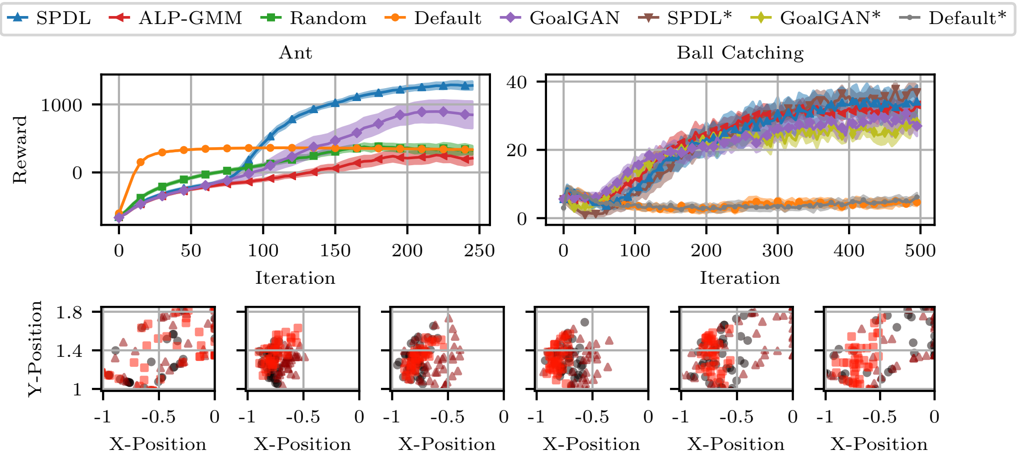


Figure 3: Mean (thick line) and two times standard error (shaded area) of the reward achieved with different curricula in the Ant environment for PPO and in the Ball-Catching environment for SAC (upper plots). The statistics are computed from 20 seeds. For Ball-Catching, runs of SPDL/GoalGAN with an initialized context distribution and runs of Default learning without policy initialization are indicated by asterisks. The lower plots show ball positions in the 'catching' plane sampled from the context distributions $p(\boldsymbol{c})$ in the Ball-Catching environment at iterations 0, 50, 80, 110, 150 and 200 (from left to right). Different sample colors and shapes indicate different algorithm runs. Given that $p(\boldsymbol{c})$ is initialized with $\mu(\boldsymbol{c})$, the samples in iteration 0 visualize the target distribution.

5.3 Ball-Catching Environment

Due to a sparse reward function and a broad target task distribution, this final environment is drastically different from the previous ones. In this environment, the agent needs to control a Barrett WAM robot to catch a ball thrown towards it. The reward function is sparse, only rewarding the robot when it catches the ball and penalizing it for excessive movements. In the simulated environment, the ball is said to be caught if it is in contact with the end effector that is attached to the robot. The context $\boldsymbol{c} \in \mathbb{R}^3$ parameterizes the distance to the robot from which the ball is thrown as well as its target position in a plane that intersects the base of the robot. Figure 1 shows the robot as well as the target distribution over the ball positions in the aforementioned 'catching' plane. In this environment, the context $\boldsymbol{c}$ is not visible to the policy, as it only changes the initial state distribution $p(\boldsymbol{s}_0)$ via the encoded target position and initial distance to the robot. Given that the initial state is already observed by the policy, observing the context is superfluous. To tackle this learning task with a curriculum, we initialize the policy of the RL algorithms to hold the robot's initial position. This creates a subspace in the context space in which the policy already performs well, i.e. where the target position of the ball coincides with the initial end effector position. This can be leveraged by CL algorithms.

Since SPDL and GoalGAN support to specify the initial context distribution, we investigate whether this feature can be exploited by choosing the initial context distribution to encode the aforementioned tasks in which the initial policy performs well. When directly learning on the target context distribution without a curriculum, it is not clear whether the policy initialization benefits learning. Hence, we evaluate the performance both with and without a pre-trained policy when not using a curriculum.

Figure 3 and Table 1 show the performance of the investigated curriculum learning approaches. We see that sampling tasks directly from the target distribution does not allow the agent to learn a meaningful policy, regardless of the initial one. Further, all curricula enable learning in this environment and achieve a similar reward. The results also highlight that initialization of the context distribution does not significantly change the performance in this task. The context distributions $p(\boldsymbol{c})$ visualized in Figure 3 indicate that SPDL shrinks the initially wide context distribution in early iterations to recover the subspace of ball target positions, in which the initial policy performs well. From there, the context distribution then gradually matches the target one. As in the point mass experiment, this happens with differing pace, as can be seen in the visualizations of $p(\boldsymbol{c})$ in Figure 3 for iteration 200: Two of the three distributions fully match the target distribution while the third only covers half of it.

6 Conclusion

We proposed self-paced deep reinforcement learning, an inference-derived curriculum reinforcement learning algorithm. The resulting method is easy to use, allows to draw connections to established regularization techniques for inference, and generalizes previous results in the domain of CRL. In our experiments, the method matched or surpassed performance of other CRL algorithms, especially excelling in tasks where learning is aimed at a single target task.

As discussed, the inference view provides many possibilities for future improvements of the proposed algorithm, such as using more elaborate methods for choosing the hyperparameter α or approximating the variational distribution $q(\boldsymbol{c})$ using more advanced methods. Such algorithmic improvements are expected to further improve the efficiency of the algorithm. Furthermore, a re-interpretation of the self-paced learning algorithm for supervised learning tasks using the presented inference perspective may allow for a unifying view across the boundary of both supervised- and reinforcement learning, allowing to share algorithmic advances.

Broader Impact

This work proposed a method to speed up and stabilize the learning of autonomous agents via curriculum reinforcement learning. In a practical scenario, such methods can reduce the amount of time, energy, or manual labor required to create autonomous agents for a given task, allowing for economic benefits. Given the inherent goal of RL to create versatile learning algorithms, free of ties to a specific domain, RL algorithms can be used in a variety of fields, ranging from automating aspects of elderly care over autonomous vehicles to military uses. Given the abstract nature of our work, it is, however, hard to estimate the immediate consequences of our work on society, since the algorithmic benefits arising from our work apply equally to all of the aforementioned examples.

Acknowledgments and Disclosure of Funding

This project has received funding from the DFG project PA3179/1-1 (ROBOLEAP). During parts of the work on this paper, Joni Pajarinen was affiliated with Tampere University, Finland.

References

[1] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.

[2] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[3] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[4] Marlos C Machado, Marc G Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *AAAI*, 2020.

[5] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *NIPS*, 2017.

[6] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *NIPS*, 2016.

[7] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *NIPS*, 2016.

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009.

[9] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *ICML*, 2018.

[10] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *CoRL*, 2017.

[11] M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *NIPS*, 2010.

[12] Pascal Klink, Hany Abdulsamad, Boris Belousov, and Jan Peters. Self-paced contextual reinforcement learning. In *CoRL*, 2019.

[13] Zhipeng Ren, Daoyi Dong, Huaxiong Li, and Chunlin Chen. Self-paced prioritized curriculum learning with coverage penalty in deep reinforcement learning. *IEEE transactions on neural networks and learning systems*, 29(6):2216–2226, 2018.

[14] Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *ICML*, 2006.

[15] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[16] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[19] Burrhus Frederic Skinner. *The behavior of organisms: An experimental analysis*. BF Skinner Foundation, 2019.

[20] J Bongard and Hod Lipson. Once more unto the breach: Co-evolving a robot and its simulator. In *ALIFE*, 2004.

[21] Tom Erez and William D Smart. What does shaping mean for computational reinforcement learning? In *ICDL*, 2008.

[22] Sanmit Narvekar and Peter Stone. Learning curriculum policies for reinforcement learning. In *AAMAS*, 2019.

[23] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *ICML*, 2018.

[24] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NIPS*, 2017.

[25] Jürgen Schmidhuber. Curious model-building control systems. In *IJCNN*, 1991.

[26] Adrien Baranes and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration for active motor learning in robots: A case study. In *IROS*, 2010.

[27] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *CoRL*, 2019.

[28] Pierre Fournier, Olivier Sigaud, Mohamed Chetouani, and Pierre-Yves Oudeyer. Accuracy-based curriculum learning in deep reinforcement learning. *arXiv preprint arXiv:1806.09614*, 2018.

[29] Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2): 265–286, 2007.

[30] Douglas Blank, Deepak Kumar, Lisa Meeden, and James B Marshall. Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture. *Cybernetics and Systems: An International Journal*, 36(2):125–150, 2005.

[31] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G Hauptmann. Self-paced curriculum learning. In *AAAI*, 2015.

[32] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. Self-paced learning with diversity. In *NIPS*, 2014.

[33] Peter Dayan and Geoffrey E Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

[34] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[35] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *IJCAI*, 2013.

[36] Philipp Hennig, Michael A Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2179):20150142, 2015.

[37] Simon JD Prince. *Computer vision: models, learning, and inference*. Cambridge University Press, 2012.

[38] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

[39] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *AAAI*, 2010.

[40] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *ICLR*, 2018.

[41] Matthew Fellows, Anuj Mahajan, Tim GJ Rudner, and Shimon Whiteson. Virel: A variational inference framework for reinforcement learning. In *NIPS*, 2019.

[42] Chang Liu, Jingwei Zhuo, Pengyu Cheng, Ruiyi Zhang, Jun Zhu, and Lawrence Carin. Understanding and accelerating particle-based variational inference. *ICML*, 2019.

[43] Andre Wibisono. Sampling as optimization in the space of measures: The langevin dynamics as a composite optimization problem. *COLT*, 2018.

[44] Stephan Mandt, James McInerney, Farhan Abrol, Rajesh Ranganath, and David Blei. Variational tempering. In *AISTATS*, 2016.

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

[46] Aditya Modi, Nan Jiang, Satinder Singh, and Ambuj Tewari. Markov decision processes with continuous side information. In *ALT*, 2018.

[47] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *ICML*, 2015.

[48] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *AAMAS*, 2011.

[49] Kentaro Katahira, Kazuho Watanabe, and Masato Okada. Deterministic annealing variant of variational bayes method. In *Journal of Physics: Conference Series*, volume 95, page 012015. IOP Publishing, 2008.

[50] Naonori Ueda and Ryohei Nakano. Deterministic annealing variant of the em algorithm. In *NIPS*, 1995.

[51] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[52] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[53] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[54] Nvidia. Isaac gym. `https://developer.nvidia.com/gtc/2019/video/S9918`, 2019. Accessed: 2020-02-06.

[55] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

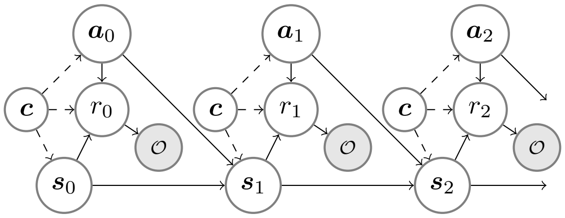


Figure 4: The extended graphical model used for the presented CRL algorithm. Solid lines mark connections that are present in the 'single task' RL problem. The dashed lines represent the additional connections that occur in the contextual RL setting, where a contextual variable $\boldsymbol{c}$ influences the MDP. Note that $\boldsymbol{c}$ and $\mathcal{O}$ refer to the same variables across all timesteps.

A Proofs

We start by restating the Latent-Variable Model for the Contextual RL setting

$$p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O}) = \int p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O},\tau,\boldsymbol{c})d\tau d\boldsymbol{c} \propto \int f(R(\tau,\boldsymbol{c}))p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c})p_{\boldsymbol{\nu}}(\boldsymbol{c})d\tau d\boldsymbol{c}, \tag{7}$$

where $p(\mathcal{O}|\tau,\boldsymbol{c}) \propto f(R(\tau,\boldsymbol{c}))$ with $R(\tau,\boldsymbol{c}) = \sum_{t\geq 0} r_{\boldsymbol{c}}(\boldsymbol{s}_t,\boldsymbol{a}_t)$ and the monotonic transformation $f:\mathbb{R}\mapsto\mathbb{R}_{\geq 0}$ defines the probability of trajectory τ being optimal in context $\boldsymbol{c}$. In LVM (7), $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ is the distribution over contexts and $p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c})$ is the probability of a trajectory, that depends on the policy $\pi_{\boldsymbol{\omega}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{c})$

$$p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c})=p_{0,\boldsymbol{c}}(\boldsymbol{s}_0)\prod_{t\geq 0}\bar{p}_{\boldsymbol{c}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)\pi_{\boldsymbol{\omega}}(\boldsymbol{a}_t|\boldsymbol{s}_t,\boldsymbol{c}). \tag{8}$$

Please note the distribution $\bar{p}_{\boldsymbol{c}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)$. This modified version of the original transition dynamics $p_{\boldsymbol{c}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)$ is used to account for the discouting factor $\gamma \leq 1$ that is present in the infinite horizon MDP setting. The dynamics $\bar{p}_{\boldsymbol{c}}$ are defined by introducing a terminal state $\boldsymbol{s}_T$, with $r(\boldsymbol{s}_T,\boldsymbol{a}) = 0$ for all $\boldsymbol{a}\in\mathcal{A}$, to which a transition can occur from any state with probability $1-\gamma$

$$\bar{p}_{\boldsymbol{c}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t) = \begin{cases} 1, & \text{if } \boldsymbol{s}_t = \boldsymbol{s}_T \text{ and } \boldsymbol{s}_{t+1} = \boldsymbol{s}_T \\ 0, & \text{if } \boldsymbol{s}_t = \boldsymbol{s}_T \text{ and } \boldsymbol{s}_{t+1} \neq \boldsymbol{s}_T \\ (1-\gamma), & \text{if } \boldsymbol{s}_t \neq \boldsymbol{s}_T \text{ and } \boldsymbol{s}_{t+1} = \boldsymbol{s}_T \\ \gamma p_{\boldsymbol{c}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t), & \text{else.} \end{cases}$$

Figure 4 visualizes the structure of LVM (7). The term Latent-Variable Model arises because, conceptually, we think about states, actions and contexts as being 'hidden'. This means that there is an underlying distribution over states, actions and contexts, which is, however, marginalized out, leaving only the quantity of interest - the 'optimality' event $\mathcal{O}$. This marginalization makes direct optimization of likelihood (7) intractable. The EM algorithm [55], as applied in the main paper, introduces a variational distribution $q(\boldsymbol{c})$ which decomposes the logarithm of likelihood (7)

$$\log\left(p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O})\right) = \int q(\boldsymbol{c})\log\left(p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O})\right)d\boldsymbol{c} \tag{9}$$

$$= \int q(\boldsymbol{c})\log\left(\frac{q(\boldsymbol{c})}{q(\boldsymbol{c})}\frac{p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O},\boldsymbol{c})}{p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})}\right)d\boldsymbol{c} \tag{10}$$

$$=E_{q(\boldsymbol{c})}\left[\log\left(\frac{p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O},\boldsymbol{c})}{q(\boldsymbol{c})}\right)\right] + D_{\text{KL}}\left(q(\boldsymbol{c})\|p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})\right). \tag{11}$$

The reformulation of the marginal likelihood $p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O})$ between lines (9) and (10) is possible because $p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O},\boldsymbol{c}) = p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O})$. Decomposing the likelihood is beneficial, since it allows to split the optimization of $\log\left(p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O})\right)$ into two steps that can be tackled individually, the so called E- and M-Step. The E-Step minimizes the second term in Eq. (11), yielding $q(\boldsymbol{c}) = p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})$ if $q(\boldsymbol{c})$ is not restricted to a parametric form. The M-Step then maximizes the first term of Eq. (11) w.r.t. $\boldsymbol{\nu}$.

Before proving our first result from the main paper, we quickly note that for our model, the M-Step can be equally thought of as minimizing $D_{\text{KL}}\left(q(\boldsymbol{c})\|p_{\boldsymbol{\nu}}(\boldsymbol{c})\right)$ w.r.t. $\boldsymbol{\nu}$, since

$$E_{q(\boldsymbol{c})}\left[\log\left(\frac{p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O},\boldsymbol{c})}{q(\boldsymbol{c})}\right)\right] = E_{q(\boldsymbol{c})}\left[\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})\right)\right] - E_{q(\boldsymbol{c})}\left[\log\left(\frac{q(\boldsymbol{c})}{p_{\boldsymbol{\nu}}(\boldsymbol{c})}\right)\right], \tag{12}$$

where the first term is constant w.r.t. $\boldsymbol{\nu}$ and the second term is equal to $-D_{\text{KL}}\left(q(\boldsymbol{c})\|p_{\boldsymbol{\nu}}(\boldsymbol{c})\right)$.

A.1 Theorem 1

This theorem establishes the connection between the maximization of our proposed objective for Curriculum generation w.r.t. $\boldsymbol{\nu}$

$$\max_{\boldsymbol{\nu}} J(\boldsymbol{\nu}, \boldsymbol{\omega}) - \alpha D_{\text{KL}}\left(p_{\boldsymbol{\nu}}(\boldsymbol{c}) \| \mu(\boldsymbol{c})\right), \quad \alpha \geq 0 \tag{13}$$

and the discussed EM algorithm when applied to LVM 7. More precisely, we show that modifications of the E-Step allow to relate Objective (13) to the execution of the E- and M-Step.

Theorem 1. *Choosing $f(\cdot){=}\exp(\cdot)$, maximizing Objective (13) minus a KL divergence term $D_{KL}\left(p_{\boldsymbol{\nu}}(\boldsymbol{c}) \| p_{\tilde{\boldsymbol{\nu}}}(\boldsymbol{c})\right)$ is equal to executing E- and M-Step while restricting $q(\boldsymbol{c})$ to be of the same parametric form as $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ and introducing a regularized E-Step $D_{KL}\left(q(\boldsymbol{c}) \middle\| \frac{1}{Z} p_{\tilde{\boldsymbol{\nu}}, \boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}}\right)$.*

Proof. As we restrict $q(\boldsymbol{c})$ to be of the same parametric form as $p_{\boldsymbol{\nu}}(\boldsymbol{c})$, an M-Step becomes superfluous, because the optimal solution of this M-Step clearly matches $q(\boldsymbol{c})$. We see that, when restricting $q(\boldsymbol{c})$ to the same parametric form as $p_{\boldsymbol{\nu}}(\boldsymbol{c})$, executing E- and M-Step is equal to simply minimzing the E-Step, where $q(\boldsymbol{c})$ is replaced by $p_{\boldsymbol{\nu}}(\boldsymbol{c})$

$$\min_{\boldsymbol{\nu}} D_{\text{KL}}\left(p_{\boldsymbol{\nu}}(\boldsymbol{c}) \middle\| \frac{1}{Z} p_{\tilde{\boldsymbol{\nu}}, \boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}}\right). \tag{14}$$

Consequently, we are left to show that above optimization problem is the same as the maximization of Objective 13. This is, however, a task of simple reformulation

$$\min_{\boldsymbol{\nu}} D_{\text{KL}}\left(p_{\boldsymbol{\nu}}(\boldsymbol{c}) \middle\| \frac{1}{Z} p_{\tilde{\boldsymbol{\nu}}, \boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}}\right) \tag{15}$$

$$= \min_{\boldsymbol{\nu}} Z + E_{p_{\boldsymbol{\nu}}(\boldsymbol{c})}\left[\log\left(\frac{p_{\boldsymbol{\nu}}(\boldsymbol{c})}{p_{\tilde{\boldsymbol{\nu}}, \boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}}}\right)\right] \tag{16}$$

$$= \max_{\boldsymbol{\nu}} -Z + \frac{1}{1+\alpha} E_{p_{\boldsymbol{\nu}}(\boldsymbol{c})}\left[\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})\right)\right] + \frac{1}{1+\alpha} p_{\tilde{\boldsymbol{\nu}}, \boldsymbol{\omega}}(\mathcal{O}) \tag{17}$$

$$- E_{p_{\boldsymbol{\nu}}(\boldsymbol{c})}\left[\log\left(\frac{p_{\boldsymbol{\nu}}(\boldsymbol{c})}{p_{\tilde{\boldsymbol{\nu}}}(\boldsymbol{c})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}}}\right)\right]. \tag{18}$$

Before proceeding to reformulate above KL-Divergence, we note that we can simply remove the normalization constant Z as well as the term $\frac{1}{1+\alpha} p_{\tilde{\boldsymbol{\nu}}, \boldsymbol{\omega}}(\mathcal{O})$, since they are constant w.r.t. $\boldsymbol{\nu}$. Furthermore, we can rescale the objective by $1 + \alpha$ without changing the optimal solution, yielding

$$\max_{\boldsymbol{\nu}} E_{p_{\boldsymbol{\nu}}(\boldsymbol{c})}\left[\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})\right)\right] - (1+\alpha) E_{p_{\boldsymbol{\nu}}(\boldsymbol{c})}\left[\log\left(\frac{p_{\boldsymbol{\nu}}(\boldsymbol{c})}{p_{\tilde{\boldsymbol{\nu}}}(\boldsymbol{c})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}}}\right)\right] \tag{19}$$

$$= \max_{\boldsymbol{\nu}} E_{p_{\boldsymbol{\nu}}(\boldsymbol{c})}\left[\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})\right)\right] - D_{\text{KL}}\left(p_{\boldsymbol{\nu}}(\boldsymbol{c}) \| p_{\tilde{\boldsymbol{\nu}}}(\boldsymbol{c})\right) - \alpha D_{\text{KL}}\left(p_{\boldsymbol{\nu}}(\boldsymbol{c}) \| \mu(\boldsymbol{c})\right). \tag{20}$$

The last reformulation was possible since we can write $p_{\boldsymbol{\nu}}(\boldsymbol{c}) = p_{\boldsymbol{\nu}}(\boldsymbol{c})^{\frac{1}{1+\alpha}} p_{\boldsymbol{\nu}}(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}}$. To proof Theorem 1, we simply need to relate the quantity $E_{p_{\boldsymbol{\nu}}(\boldsymbol{c})}\left[\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})\right)\right]$ to $J(\boldsymbol{\nu}, \boldsymbol{\omega})$. Using $f(R(\tau, \boldsymbol{c})) = \exp(R(\tau, \boldsymbol{c}))$ and Jensens inequality, we can show that

$$\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})\right) = \log\left(\int \exp(R(\tau, \boldsymbol{c})) p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c}) d\tau\right) - \log(Z) \tag{21}$$

$$\geq \int R(\tau, \boldsymbol{c}) p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c}) d\tau - \log(Z) \tag{22}$$

$$= \int \sum_{t \geq 0} r(\boldsymbol{s}_t, \boldsymbol{a}_t) p_{0,\boldsymbol{c}}(\boldsymbol{s}_0) \prod_{t \geq 0} \bar{p}_{\boldsymbol{c}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) \pi_{\boldsymbol{\omega}}(\boldsymbol{a}_t|\boldsymbol{s}_t, \boldsymbol{c}) d\boldsymbol{s}_t d\boldsymbol{a}_t - \log(Z) \tag{23}$$

$$= \int \sum_{t \geq 0} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) p_{0,\boldsymbol{c}}(\boldsymbol{s}_0) \prod_{t \geq 0} p_{\boldsymbol{c}}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) \pi_{\boldsymbol{\omega}}(\boldsymbol{a}_t|\boldsymbol{s}_t, \boldsymbol{c}) d\boldsymbol{s}_t d\boldsymbol{a}_t - \log(Z) \tag{24}$$

$$= E_{p_{0,\boldsymbol{c}}(\boldsymbol{s})}\left[V_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{c})\right] - \log(Z) \tag{25}$$

The reformulation between lines (23) and (24) is possible because of the modified dynamics. The chance of not transitioning into $\boldsymbol{s}_T$ for t steps is given by γ^t. Since the agent recieves no reward in $\boldsymbol{s}_T$, any terms of the form $r(\boldsymbol{s}_T, \boldsymbol{a}_t)$ can be removed from the expectation in line (23). Combining these two observations yields line (24). Given that the normalization constant Z is constant across all contexts $\boldsymbol{c}$, we can again remove it from the optimization of the reformulated E-Step (Eq. 20). With that we see that when choosing $f(R(\tau, \boldsymbol{c})) = \exp(R(\tau, \boldsymbol{c}))$, it holds that $E_{p_{\boldsymbol{\nu}}(\boldsymbol{c})}\left[\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})\right)\right] \geq J(\boldsymbol{\nu}, \boldsymbol{\omega})$. Consequently, we optimize the E-Step using a lower bound by optimizing Objective 13. Given that we can skip the M-Step due to restricting the form of $q(\boldsymbol{c})$, we see that we are indeed performing the steps of the EM algorithm outlined in Theorem 1. □

A.2 Theorem 2

This theorem shows that the update rule for the context distribution, established by Klink et al. [12], is also explained as applying EM to maximize $p_{\boldsymbol{\nu},\boldsymbol{\omega}}(\mathcal{O})$ w.r.t. $\boldsymbol{\nu}$. In this case, however, the variational distribution is not restricted to a parametric form, requiring an explicit M-Step. Looking at the work by Klink et al. [12], we see that their algorithm indeed performs an M-Step by fitting a parametric model to weighted samples (which approximately represent $q(\boldsymbol{c})$).

Theorem 2. *Choosing* $f(\cdot) = \exp(\cdot/\eta)$*, the (unrestricted) variational distribution after the regularized E-Step is given by* $q(\boldsymbol{c}) \propto p_{\boldsymbol{\nu}}(\boldsymbol{c}) \exp\left(\frac{V_{\boldsymbol{\omega}}(\boldsymbol{c}) + \eta\alpha(\log(\mu(\boldsymbol{c})) - \log(p_{\boldsymbol{\nu}}(\boldsymbol{c})))}{\eta + \eta\alpha}\right)$*, where* $V_{\boldsymbol{\omega}}(\boldsymbol{c})$ *is the 'episodic value function' as defined in [34].*

Proof. We first note that, given that we are not restricting $q(\boldsymbol{c})$ to any parametric form, $q(\boldsymbol{c}) = \frac{1}{Z} p_{\tilde{\boldsymbol{\nu}},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}}$ holds after the E-Step. A reformulation of this probabilitty distribution brings us closer to the desired result

$$\frac{1}{Z} p_{\tilde{\boldsymbol{\nu}},\boldsymbol{\omega}}(\boldsymbol{c}|\mathcal{O})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}} \tag{26}$$

$$\propto p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})^{\frac{1}{1+\alpha}} p_{\boldsymbol{\nu}}(\boldsymbol{c})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}} \tag{27}$$

$$\propto p_{\boldsymbol{\nu}}(\boldsymbol{c}) p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}} p_{\boldsymbol{\nu}}(\boldsymbol{c})^{\frac{-\alpha}{1+\alpha}} \tag{28}$$

$$\propto p_{\boldsymbol{\nu}}(\boldsymbol{c}) \exp\left(\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})^{\frac{1}{1+\alpha}} \mu(\boldsymbol{c})^{\frac{\alpha}{1+\alpha}} p_{\boldsymbol{\nu}}(\boldsymbol{c})^{\frac{-\alpha}{1+\alpha}}\right)\right) \tag{29}$$

$$\propto p_{\boldsymbol{\nu}}(\boldsymbol{c}) \exp\left(\frac{\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})\right) + \alpha\left(\log(\mu(\boldsymbol{c})) - \log(p_{\boldsymbol{\nu}}(\boldsymbol{c}))\right)}{1+\alpha}\right) \tag{30}$$

To proof Theorem 2, we need to relate $\log\left(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})\right)$ to the 'episodic value function' $V_{\boldsymbol{\omega}}(\boldsymbol{c}) = \eta \log\left(\int \exp\left(R(\tau|\boldsymbol{c})/\eta\right) p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c}) d\tau\right)$ as defined in [34]. By choosing the transformation $f(R(\tau, \boldsymbol{c})) = \exp\left(\frac{R(\tau,\boldsymbol{c})}{\eta}\right)$, it follows that

$$\log(p_{\boldsymbol{\omega}}(\mathcal{O}|\boldsymbol{c})) \tag{31}$$

$$= \log\left(\int p(\mathcal{O}|\tau, \boldsymbol{c}) p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c}) d\tau\right) \tag{32}$$

$$\propto \log\left(\int \exp\left(\frac{R(\tau, \boldsymbol{c})}{\eta}\right) p_{\boldsymbol{\omega}}(\tau|\boldsymbol{c}) d\tau\right) = \frac{1}{\eta} V_{\boldsymbol{\omega}}(\boldsymbol{c}). \tag{33}$$

Inserting this result into Eq. (30) proofs the theorem. □

B Experimental Details

In this section, we present details that could not be included in the main paper due to space limitations. This includes parameters of the employed algorithms, additional details about the mechanics of the environments as well as a qualitative discussion of the results.

The parameters of SPDL for different environments and RL algorithms are shown in Table 2. The parameters N_α and ζ have the same meaning as in the main paper. The additional parameter n_{OFFSET} describes the number of RL algorithm iterations that take place before SPDL is allowed the change

Table 2: Hyperparameters for the SPDL algorithm per environment and RL algorithm. The asterisks in the table mark the Ball-Catching experiments with an initialized context distribution.

	N_α	ζ	n_{OFFSET}	n_{STEP}	$\boldsymbol{\sigma}_{\text{LB}}$	$D_{\text{KL}_{LB}}$
POINT-MASS (TRPO)	70	1.6	5	2048	$[0.2\ \ 0.1875\ \ 0.1]$	8000
POINT-MASS (PPO)	10	1.4	5	2048	$[0.2\ \ 0.1875\ \ 0.1]$	8000
POINT-MASS (SAC)	50	1.2	5	2048	$[0.2\ \ 0.1875\ \ 0.1]$	8000
ANT (PPO)	15	0.4	40	81920	$[1\ \ 0.5]$	11000
BALL-CATCHING (TRPO)	70	0.4	5	5000	-	-
BALL-CATCHING* (TRPO)	0	0.425	5	5000	-	-
BALL-CATCHING (PPO)	50	0.45	5	5000	-	-
BALL-CATCHING* (PPO)	0	0.45	5	5000	-	-
BALL-CATCHING (SAC)	60	0.6	5	5000	-	-
BALL-CATCHING* (SAC)	0	0.6	5	5000	-	-

the context distribution. This parameter can be necessary if some iterations are required until the approximated value function produces meaningful estimates of the expected value. In the ant environment, we realized that the agent takes a certain amount of time (roughly 40 iterations) until it manages to reach the wall. Only then, the difference in task difficulty becomes apparent. The parameter n_{OFFSET} allows to compensate for such task-specific details. This procedure corresponds to providing parameters of a pre-trained policy as $\boldsymbol{\omega}_0$ in the algorithm sketched in the main paper. We selected the best ζ for every RL algorithm by a simple grid-search in an interval around a reasonably working parameter that was found by simple trial and error. For the PointMass environment, we only tuned the hyperparameters for SPDL in the experiment with a three-dimensional context space and reused them for the two-dimensional context space. To conduct the experiments, we use the implementation of ALP-GMM, GoalGAN and SPRL provided in the repositories accompanying the papers [23, 27, 12].

For ALP-GMM we tuned the percentage of random samples drawn from the context space p_{RAND}, the number of policy rollouts between the update of the context distribution n_{ROLLOUT} as well as the maximum buffer size of past trajectories to keep s_{BUFFER}. For each environment and algorithm, we did a grid-search over

$$(p_{\text{RAND}}, n_{\text{ROLLOUT}}, s_{\text{BUFFER}}) \in \{0.1, 0.2, 0.3\} \times \{50, 100, 200\} \times \{500, 1000, 2000\}.$$

For GoalGAN we tuned the amount of random noise that is added on top of each sample δ_{NOISE}, the number of policy rollouts between the update of the context distribution n_{ROLLOUT} as well as the percentage of samples drawn from the success buffer p_{SUCCESS}. For each environment and algorithm, we did a grid-search over

$$(\delta_{\text{NOISE}}, n_{\text{ROLLOUT}}, p_{\text{SUCCESS}}) \in \{0.025, 0.05, 0.1\} \times \{50, 100, 200\} \times \{0.1, 0.2, 0.3\}.$$

The results of the hyperparameter optimization for GoalGAN and ALP-GMM are shown in Table 3.

The similarity of our algorithm and SPRL – and since we could only apply it to one experiment due to numerical reasons – allowed to start from the parameters of SPDL and obtain well-working parameters by a few adjustments.

In the experiments, we found that restricting the standard deviation of the context distribution to stay above a certain lower bound $\boldsymbol{\sigma}_{\text{LB}}$ helps to stabilize learning when generating curricula for narrow target distributions with SPDL. Although such constraints could be included rigorously via constraints on the distribution $p_{\boldsymbol{\nu}}(\boldsymbol{c})$ in the E-Step, we accomplish this by just clipping the standard deviation until the KL-Divergence w.r.t. the target distribution falls below a certain threshold $D_{\text{KL}_{\text{LB}}}$. This technique was also employed by Klink et al. [12].

Opposed to the sketched algorithm in the main paper, we specify the number of steps n_{STEP} in the environment instead of the number of trajectory rollouts between context distribution updates in our implementation.

Since for all environments, both initial- and target distribution are Gaussians with independent noise in each dimension, we specify them in Table 4 by providing their mean $\boldsymbol{\mu}$ and the vector of standard deviations for each dimension $\boldsymbol{\delta}$. When sampling from a Gaussian, the resulting context is clipped to stay in the defined context space.

Table 3: Hyperparameters for the ALP-GMM and GoalGAN algorithm per environment and RL algorithm. The abbreviation AG is used for ALP-GMM, while GG stands for GoalGAN.

	p_{RAND}	$n_{\text{ROLLOUT}_{\text{AG}}}$	s_{BUFFER}	δ_{NOISE}	$n_{\text{ROLLOUT}_{\text{GG}}}$	p_{SUCCESS}
Point-Mass 3D (TRPO)	0.1	100	1000	0.05	200	0.2
Point-Mass 3D (PPO)	0.1	100	500	0.025	200	0.1
Point-Mass 3D (SAC)	0.1	200	1000	0.1	100	0.1
Point-Mass 2D (TRPO)	0.3	100	500	0.1	200	0.2
Point-Mass 2D (PPO)	0.2	100	500	0.1	200	0.3
Point-Mass 2D (SAC)	0.2	200	1000	0.025	50	0.2
Ant (PPO)	0.1	50	500	0.05	125	0.2
Ball-Catching (TRPO)	0.2	200	2000	0.1	200	0.3
Ball-Catching (PPO)	0.3	200	2000	0.1	200	0.3
Ball-Catching (SAC)	0.3	200	1000	0.1	200	0.3

If necessary, we tuned the hyperparameters of the RL algorithms by hand on easier versions of the target task, not employing any Curriculum. The goal was to be as fair as possible by not optimizing the RL algorithm for a specific curriculum. For the Ant and PointMass environment, this was done by training on a wide gate positioned right in front of the agent. For the Ball-Catching environment, this was done by training on a version of the environment with dense reward. For PPO, we use the "PPO2" implementation of Stable-Baselines.

The experiments were conducted on a computer with an AMD Ryzen 9 3900X 12-Core Processor, an Nvidia RTX 2080 graphics card and 64GB of RAM.

B.1 Point-Mass Environment

The state of this environment is comprised of the position and velocity of the point-mass $\boldsymbol{s} = [x\ \dot{x}\ y\ \dot{y}]$. The actions correspond to the force applied in x- and y-dimension $\boldsymbol{a} = [F_x\ F_y]$. The context encodes position and width of the gate as well as the dynamic friction coefficient of the ground on which the point mass slides $\boldsymbol{c} = [p_g\ w_g\ \mu_k] \in [-4, 4] \times [0.5, 8] \times [0, 4] \subset \mathbb{R}^3$. The dynamics of the system are defined by

$$\begin{pmatrix} \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & -\mu_k & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\mu_k \end{pmatrix} \boldsymbol{s} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \boldsymbol{a}.$$

The x- and y- position of the point mass is enforced to stay within the space $[-4, 4] \times [-4, 4]$. The gate is located at position $[p_g\ 0]$. If the agent crosses the line $y = 0$, we check whether its x-position is within the interval $[p_g - 0.5w_g, p_g + 0.5w_g]$. If this is not the case, we stop the episode as the agent has crashed into the wall. Each episode is terminated after a maximum of 100 steps. The reward function is given by

$$r(\boldsymbol{s}, \boldsymbol{a}) = \exp\left(-0.6\|\boldsymbol{o} - [x\ y]\|_2\right),$$

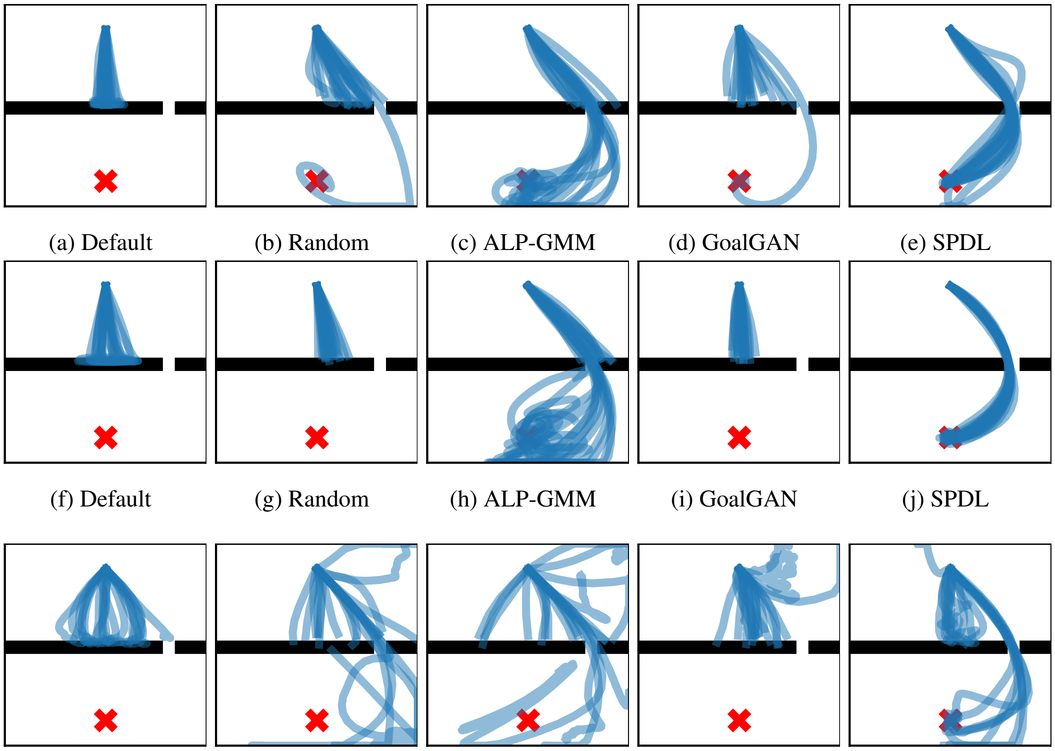
where $\boldsymbol{o} = [0\ \ -3]$, $\|\cdot\|_2$ is the L2-Norm. The agent is always initialized at state $\boldsymbol{s}_0 = [0\ 0\ 3\ 0]$.

For all RL algorithms, we use a discount factor of $\gamma = 0.95$ and represent policy and value function by networks using 21 hidden layers with tanh activations. For TRPO and PPO, we take 2048 steps in the environment between policy updates.

For TRPO we set the GAE parameter $\lambda = 0.99$, the maximum allowed KL-Divergence to 0.004 and the value function step size $a_v \approx 0.24$, leaving all other parameters to their implementation defaults.

For PPO we use GAE parameter $\lambda = 0.99$, an entropy coefficient of 0 and disable the clipping of the value function objective. The number of optimization epochs is set to 8 and we use 32 mini-batches. All other parameters are left to their implementation defaults.

For SAC, we use an experience-buffer of 10000 samples, leaving every other setting to the implementation default. Hence we use the soft Q-Updates and update the policy after every environment step.



(a) Default (b) Random (c) ALP-GMM (d) GoalGAN (e) SPDL

(f) Default (g) Random (h) ALP-GMM (i) GoalGAN (j) SPDL

(k) Default (l) Random (m) ALP-GMM (n) GoalGAN (o) SPDL

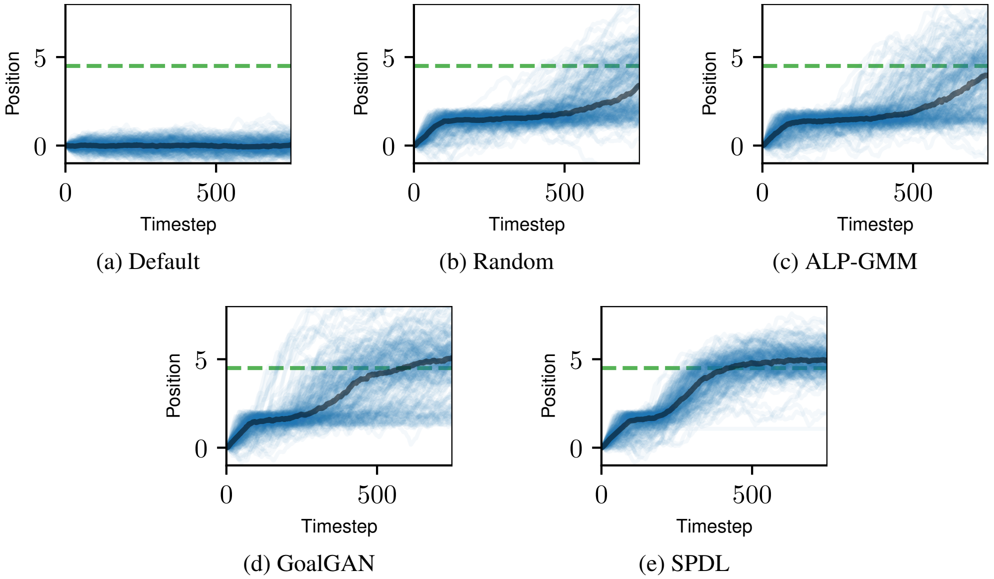
Figure 5: Visualizations of policy rollouts in the Point-Mass Environment (three context dimensions) with policies learned using different curricula and RL algorithms. Each rollout was generated using a policy learned with a different seed. The first row shows results for TRPO, the second for PPO and the third shows results for SAC.

For SPRL, we use $K_\alpha = 40$, $n_{\text{OFFSET}} = 0$, $\zeta = 2.0$ for the 3D- and $\zeta = 1.5$ and 2D case. We use the same values for σ_{LB} and $D_{\text{KL}_{\text{LB}}}$ as for SPDL (Table 2). Between updates of the episodic policy, we do 25 policy rollouts and keep a buffer containing rollouts from the past 10 iterations, resulting in 250 samples for policy- and context distribution update. The linear policy over network weights is initialized to a zero-mean Gaussian with unit variance. We use Polynomial features up to degree two for approximating the value function during policy optimization. For the allowed KL-Divergence, we observed best results when using $\epsilon = 0.5$ for the weight computation of the samples, but using a lower value of $\epsilon = 0.2$ when fitting the parametric policy to these weighted samples. We suppose that the higher value of ϵ during weight computation counteracts the effect of the buffer containing policy samples from earlier iterations.

Looking at Figure 5, we can see that ALP-GMM allowed to learn policies that sometimes are able to pass the gate. However, in other cases, the policies crashed the point mass into the wall. Opposed to this, directly training on the target task led to policies that learned to steer the point mass very close to the wall without crashing (which is unfortunately hard to see in the plot). Reinvestigating the above reward function, this explains the lower reward of ALP-GMM, GoalGAN and the randomly generated curriculum compared to directly learning on the target task, as a crash prevents the agent from accumulating positive rewards over time.

B.2 Ant Environment

As mentioned in the main paper, we simulate the ant using the Isaac Gym simulator [54]. This allows to speed up training time by parallelizing the simulation of policy rollouts on the graphics card. Since the Stable-Baselines implementation of TRPO and SAC do not support the use of vectorized



(a) Default (b) Random (c) ALP-GMM

(d) GoalGAN (e) SPDL

Figure 6: Visualizations of the x-position during policy rollouts in the Ant Environment with policies learned using different curricula. The blue lines correspond to 200 individual trajectories and the thick black line shows the median over these individual trajectories. The trajectories were generated from 20 algorithms runs, were each final policy was used to generate 10 trajectories.

environments, it is hard to combine Isaac Gym with these algorithms. Because of this reason, we decided not to run experiments with TRPO and SAC in the Ant environment.

The state $\boldsymbol{s} \in \mathbb{R}^{29}$ is defined to be the 3D-position of the ant's body, its angular and linear velocity as well as positions and velocities of the 8 joints of the ant. An action $\boldsymbol{a} \in \mathbb{R}^{8}$ is defined by the 8 torques that are applied to the ant's joints.

The context $\boldsymbol{c} = [p_g \; w_g] \in [-10, 10] \times [3, 13] \subset \mathbb{R}^2$ defines, just as in the Point-Mass environment, the position and width of the gate that the Ant needs to pass.

The reward function of the environment is computed based on the x-position of the ant's center of mass c_x in the following way

$$r(\boldsymbol{s}, \boldsymbol{a}) = 1 + 5 \exp\left(-0.5 \min(0, c_x - 4.5)^2\right) - 0.3 \|\boldsymbol{a}\|_2^2.$$

The constant 1 term was taken from the OpenAI Gym implementation to encourage the survival of the ant [53]. Compared to the OpenAI Gym environment, we set the armature value of the joints from 1 to 0 and also decrease the maximum torque from 150Nm to 20Nm, since the values from OpenAI Gym resulted in unrealistic movement behavior in combination with Isaac Gym. Nonetheless, these changes did not result in a qualitative change in the algorithm performances.

With the wall being located at position x=3, the agent needs to pass it in order to obtain the full environment reward by ensuring that $c_x >= 4.5$.

The policy and value function are represented by neural networks with two hidden layers of 64 neurons each and $\tanh$ activation functions. We use a discount factor $\gamma = 0.995$ for all algorithms, which can be explained due to the long time horizons of 750 steps. We take 81920 steps in the environment between a policy update. This was significantly sped-up by the use of the Isaac Gym simulator, which allowed to simulate 40 environments in parallel on a single GPU.

For PPO, we use an entropy coefficient of 0 and disable the clipping of the value function objective. All other parameters are left to their implementation defaults. We disable the entropy coefficient as we observed that for the Ant environment, PPO still tends to keep around $10 - 15\%$ of its initial additive noise even during late iterations.

Investigating Figure 6, we see that both SPDL and GoalGAN learn policies that allow to pass the gate. However, the policies learned with SPDL seem to be more reliable compared to the ones learned with GoalGAN. As mentioned in the main paper, ALP-GMM and a random curriculum also learn policies that navigate the ant towards the goal in order to pass it. However, the behavior is less directed and less reliable. Interestingly, directly learning on the target task results in a policy that tends to not move in order to avoid action penalties. Looking at the main paper, we see that this results in a similar reward compared to the inefficient policies learned with ALP-GMM and a random curriculum.

B.3 Ball-Catching Environment

In the final environment, the robot is controlled in joint space via the desired position for 5 of the 7 joints. We only control a subspace of all available joints, since it is not necessary for the robot to leave the ”catching” plane (defined by $x = 0$) that is intersected by each ball. The actions $\boldsymbol{a} \in \mathbb{R}^5$ are defined as the displacement of the current desired joint position. The state $\boldsymbol{s} \in \mathbb{R}^{21}$ consists of the positions and velocities of the controlled joints, their current desired positions, the current three-dimensional ball position and its linear velocity.

As previously mentioned, the reward function is sparse,

$$r(\boldsymbol{s}, \boldsymbol{a}) = 0.275 - 0.005\|\boldsymbol{a}\|_2^2 + \begin{cases} 50 + 25(\boldsymbol{n}_s \cdot \boldsymbol{v}_b)^5, & \text{if ball catched} \\ 0, & \text{else} \end{cases} \quad ,$$

only giving a meaningful reward when catching the ball and otherwise just a slight penalty on the actions to avoid unnecessary movements. In the above definition, $\boldsymbol{n}_s$ is a normal vector of the end effector surface and $\boldsymbol{v}_b$ is the linear velocity of the ball. This additional term is used to encourage the robot to align its end effector with the curve of the ball. If the end effector is e.g. a net (as assumed for our experiment), the normal is chosen such that aligning it with the ball maximizes the opening through which the ball can enter the net.

The context $c = [\phi, r, d_x] \in [0.125\pi, 0.5\pi] \times [0.6, 1.1] \times [0.75, 4] \subset \mathbb{R}^3$ controls the target ball position in the catching plane, i.e.

$$\boldsymbol{p}_{\text{des}} = [0 \quad -r\cos(\phi) \quad 0.75 + r\sin(\phi)].$$

Furthermore, the context determines the distance in x-dimension from which the ball is thrown

$$\boldsymbol{p}_{\text{init}} = [d_x \; d_y \; d_z],$$

where $d_y \sim \mathcal{U}(-0.75, -0.65)$ and $d_z \sim \mathcal{U}(0.8, 1.8)$ and $\mathcal{U}$ represents the uniform distribution. The initial velocity is then computed using simple projectile motion formulas by requiring the ball to reach $\boldsymbol{p}_{\text{des}}$ at time $t = 0.5 + 0.05d_x$. As we can see, the context implicitly controls the initial state of the environment.

The policy and value function networks for the RL algorithms have three hidden layers with 64 neurons each and $\tanh$ activation functions. We use a discount factor of $\gamma = 0.995$. The policy updates in TRPO and PPO are done after 5000 environment steps.

For SAC, a replay buffer size of $100,000$ is used. Due to the sparsity of the reward, we increase the batch size to 512. Learning with SAC starts after 1000 environment steps. All other parameters are left to their implementation defaults.

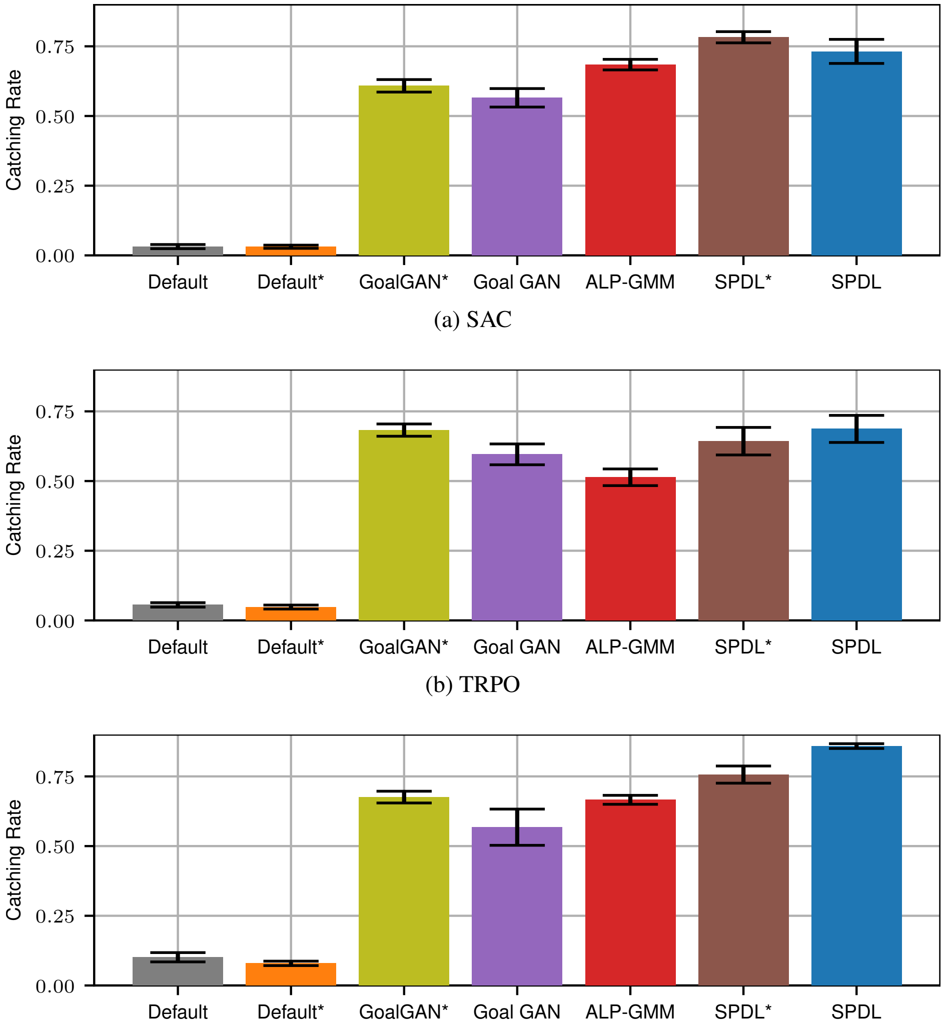
For TRPO we set the GAE parameter $\lambda = 0.95$, leaving all other parameters to their implementation defaults.

For PPO we use a GAE parameter $\lambda = 0.95$, 10 optimization epochs, 25 mini-batches per epoch, an entropy coefficient of 0 and disable the clipping of the value function objective. The remaining parameters are left to their implementation defaults.

Figure 7 visualizes the catching success rates of the learned policies. As can be seen, the performance of the policies learned with the different RL algorithms achieve comparable catching performance.

Table 4: Mean and standard deviation of target and initial distributions per environment.

	$\boldsymbol{\mu}_{\text{INIT}}$	$\boldsymbol{\delta}_{\text{INIT}}$	$\boldsymbol{\mu}_{\text{TARGET}}$	$\boldsymbol{\delta}_{\text{TARGET}}$
Point-Mass	$[0 \;\; 4.25 \;\; 2]$	$[2 \;\; 1.875 \;\; 1]$	$[2.5 \;\; 0.5 \;\; 0]$	$[0.004 \;\; 0.00375 \;\; 0.002]$
Ant	$[0 \;\; 8]$	$[3.2 \;\; 1.6]$	$[-8 \;\; 3]$	$[0.01 \;\; 0.005]$
Ball-Catching	$[0.68 \;\; 0.9 \;\; 0.85]$	$[0.03 \;\; 0.03 \;\; 0.3]$	$[1.06 \;\; 0.85 \;\; 2.375]$	$[0.8 \;\; 0.38 \;\; 1]$



(a) SAC

(b) TRPO

(c) PPO

Figure 7: Mean Catching Rate of the final policies learned with different curricula and RL algorithms on the Ball Catching environment. The mean is computed from 20 algorithm runs with different seeds. For each run, the success rate is computed from 200 ball-throws. The bars visualize the estimated standard error.

Interestingly, SAC performs comparable in terms of catching performance, although the average reward of the final policies learned with SAC is lower. This is to be credited to excessive movement and/or bad alignment of the end effector with the velocity vector of the ball.