Klink, Pascal; Abdulsamad, Hany; Belousov, Boris; D'Eramo, Carlo; Peters, Jan; Pajarinen, Joni

A probabilistic interpretation of self-paced learning with applications to reinforcement learning

Published in:
Journal of Machine Learning Research

Published: 01/07/2021

Document Version
Publisher's PDF, also known as Version of record

Journal of Machine Learning Research 22 (2021) 1-52 Submitted 2/21; Revised 6/21; Published 7/21

A Probabilistic Interpretation of Self-Paced Learning with Applications to Reinforcement Learning

Pascal Klink PASCAL.KLINK@TU-DARMSTADT.DE
Intelligent Autonomous Systems, TU Darmstadt, Germany

Hany Abdulsamad HANY.ABDULSAMAD@TU-DARMSTADT.DE
Intelligent Autonomous Systems, TU Darmstadt, Germany

Boris Belousov BORIS.BELOUSOV@TU-DARMSTADT.DE
Intelligent Autonomous Systems, TU Darmstadt, Germany

Carlo D'Eramo CARLO.DERAMO@TU-DARMSTADT.DE
Intelligent Autonomous Systems, TU Darmstadt, Germany

Jan Peters JAN.PETERS@TU-DARMSTADT.DE
Intelligent Autonomous Systems, TU Darmstadt, Germany

Joni Pajarinen JONI.PAJARINEN@AALTO.FI
Intelligent Autonomous Systems, TU Darmstadt, Germany
Department of Electrical Engineering and Automation, Aalto University, Finland

Editor: George Konidaris

Abstract

Across machine learning, the use of curricula has shown strong empirical potential to improve learning from data by avoiding local optima of training objectives. For reinforcement learning (RL), curricula are especially interesting, as the underlying optimization has a strong tendency to get stuck in local optima due to the exploration-exploitation trade-off. Recently, a number of approaches for an automatic generation of curricula for RL have been shown to increase performance while requiring less expert knowledge compared to manually designed curricula. However, these approaches are seldomly investigated from a theoretical perspective, preventing a deeper understanding of their mechanics. In this paper, we present an approach for automated curriculum generation in RL with a clear theoretical underpinning. More precisely, we formalize the well-known self-paced learning paradigm as inducing a distribution over training tasks, which trades off between task complexity and the objective to match a desired task distribution. Experiments show that training on this induced distribution helps to avoid poor local optima across RL algorithms in different tasks with uninformative rewards and challenging exploration requirements.

Keywords: curriculum learning, reinforcement learning, self-paced learning, tempered inference, rl-as-inference

1. Introduction

Research on reinforcement learning (RL) (Sutton and Barto, 1998) has led to recent successes in long-horizon planning (Mnih et al., 2015; Silver et al., 2017) and robot control (Kober and Peters, 2009; Levine et al., 2016). A driving factor of these successes has been the combination of RL paradigms with powerful function approximators, commonly referred

to as deep RL (DRL). While DRL has considerably pushed the boundary w.r.t. the type and size of tasks that can be tackled, its algorithms suffer from high sample complexity. This can lead to poor performance in scenarios where the demand for samples is not satisfied. Furthermore, crucial challenges such as poor exploratory behavior of RL agents are still far from being solved, resulting in a large body of research that aims to reduce sample complexity by improving this exploratory behavior of RL agents (Machado et al., 2020; Tang et al., 2017; Bellemare et al., 2016; Houthooft et al., 2016; Schultheis et al., 2020).
Another approach to making more efficient use of samples is to leverage similarities between learning environments and tasks in the framework of contextual- or multi-task RL. In these frameworks, a shared task structure permits simultaneous optimization of a policy for multiple tasks via inter- and extrapolation (Kupcsik et al., 2013; Schaul et al., 2015; Jaderberg et al., 2017), resulting in tangible speed ups in learning across tasks. Such approaches expose the agent to tasks drawn from a distribution under which the agent should optimize its behavior. Training on such a fixed distribution, however, does not fully leverage the contextual RL setting in case there is a difference in difficulty among tasks. In such a scenario, first training on "easier" tasks and exploiting the generalizing behavior of the agent to gradually progress to "harder" ones promises to make more efficient use of environment interaction. This idea is at the heart of curriculum learning (CL), a term introduced by Bengio et al. (2009) for supervised learning problems. By now, applications of CL have increasingly expanded to RL problems, where the aim is to design task sequences that maximally benefit the learning progress of an RL agent (Narvekar et al., 2020).
Recently, an increasing number of algorithms for an automated generation of curricula have been proposed (Baranes and Oudeyer, 2010; Florensa et al., 2017; Andrychowicz et al., 2017; Riedmiller et al., 2018). While empirically demonstrating their beneficial effect on the learning performance of RL agents, the heuristics that guide the generation of the curriculum are, as of now, theoretically not well understood. In contrast, in supervised learning, self-paced learning (Kumar et al., 2010) is an approach to curriculum generation that enjoys wide adaptation in practice (Supancic and Ramanan, 2013; Fan et al., 2018; Jiang et al., 2014a) and has a firm theoretical interpretation as a majorize-minimize algorithm applied to a regularized objective (Meng et al., 2017). In this paper, we develop an interpretation of self-paced learning as the process of generating a sequence of distributions over samples. We use this interpretation to transfer the concept of self-paced learning to RL problems, where the resulting approach generates a curriculum based on two quantities: the value function of the agent (reflecting the task complexity) and the KL divergence to a target distribution of tasks (reflecting the incorporation of desired tasks).
Contribution We propose an interpretation of the self-paced learning algorithm from a probabilistic perspective, in which the weighting of training samples corresponds to a sampling distribution (Section 4). Based on this interpretation, we apply self-paced learning to the contextual RL setting, obtaining a curriculum over RL tasks that trades-off agent performance and matching a target distribution of tasks (Section 5). We connect the approach to the RL-as-inference paradigm (Toussaint and Storkey, 2006; Levine, 2018), recovering well-known regularization techniques in the inference literature (Section 9). We experimentally evaluate algorithmic realizations of the curriculum in both episodic- (Section 6) and step-based RL settings (Section 7). Empirical evidence suggests that the scheme can match

and surpass state-of-the-art CL methods for RL in environments of different complexity and with sparse and dense rewards.

2. Related Work

Simultaneously evolving the learning task with the learner has been investigated in a variety of fields ranging from behavioral psychology (Skinner, 1938) to evolutionary robotics (Bongard and Lipson, 2004) and RL (Asada et al., 1996; Erez and Smart, 2008; Wang et al., 2019). For supervised learning, this principle was given the name *curriculum learning* by Bengio et al. (2009). The name has by now also been established in the reinforcement learning (RL) community, where a variety of algorithms aiming to generate curricula that maximally benefit the learner have been proposed.

A driving principle behind curriculum reinforcement learning (CRL) is the idea of transferring successful behavior from one task to another, deeply connecting it to the problem of transfer learning (Pan and Yang, 2009; Taylor and Stone, 2009; Lazaric, 2012). In general, transferring knowledge is—depending on the scenario—a challenging problem on its own, requiring a careful definition of what is to be transferred and what are the assumptions about the tasks between which to transfer. Aside from this problem, Narvekar and Stone (2019) showed that learning to create an *optimal* curriculum can be computationally harder than learning the solution for a task from scratch. Both of these factors motivate research on tractable approximations to the problem of transfer and curriculum generation.

To ease the problem of transferring behavior between RL tasks, a shared state-action space between tasks as well as an additional variable encoding the task to be solved are commonly assumed. This variable is usually called a goal (Schaul et al., 2015) or a context (Modi et al., 2018; Kupcsik et al., 2013). In this paper, we will adapt the second name, also treating the word "context" and "task" interchangeably, i.e. treating the additional variable and the task that it represents as the same entity.

It has been shown that function approximators can leverage the shared state-action space and the additional task information to generalize important quantities, such as value functions, across tasks (Schaul et al., 2015). This approach circumvents the complicated problem of transfer in its generality, does however impose assumptions on the set of Markov decision processes (MDPs) as well as the contextual variable that describes them. Results from Modi et al. (2018) suggest that one such assumption may be a gradual change in reward and dynamics of the MDP w.r.t. the context, although this requirement would need to be empirically verified. For the remainder of this document, we will disregard this important problem and focus on RL problems with similar characteristics as the ones investigated by Modi et al. (2018), as often done for other CRL algorithms. A detailed study of these assumptions and their impact on CRL algorithms is not known to us but is an interesting endeavor. We now continue to highlight some CRL algorithms and refer to the survey by Narvekar et al. (2020) for an extensive overview.

The majority of CRL methods can be divided into three categories w.r.t. the underlying concept. On the one hand, in tasks with binary rewards or success indicators, the idea of keeping the agent's success rate within a certain range has resulted in algorithms with drastically improved sample efficiency (Florensa et al., 2018, 2017; Andrychowicz et al., 2017). On the other hand, many CRL methods (Schmidhuber, 1991; Baranes and Oudeyer, 2010;

Portelas et al., 2019; Fournier et al., 2018) are inspired by the idea of 'curiosity' or 'intrinsic motivation' (Oudeyer et al., 2007; Blank et al., 2005)—terms that refer to the way humans organize autonomous learning even in the absence of a task to be accomplished. The third category includes algorithms that use the value function to guide the curriculum. While similar to methods based on success indicators in sparse reward settings, these methods can allow to incorporate the richer feedback available in dense rewards settings. To the best of our knowledge, only our work and that of Wöhlke et al. (2020) fall into this category. The work of Wöhlke et al. (2020) defines a curriculum over starting states using the gradient of the value function w.r.t. the starting state. The proposed curriculum prefers starting states with a large gradient norm of the value function, creating similarities to metrics used in intrinsic motivation. In our method, the value function is used as a competence measure to trade-off between easy tasks and tasks that are likely under a target distribution.

Our approach to curriculum generation builds upon the idea of *self-paced learning* (SPL), initially proposed by Kumar et al. (2010) for supervised learning tasks and extended by Jiang et al. (2014b, 2015) to allow for user-chosen penalty functions and constraints. SPL generates a curriculum by trading-off between exposing the learner to all available training samples and selecting samples in which the learner performs well. The approach has been employed in a variety of supervised learning problems (Supancic and Ramanan, 2013; Fan et al., 2018; Jiang et al., 2014a). Furthermore, Meng et al. (2017) proposed a theoretical interpretation of SPL, identifying it as a majorize-minimize algorithm applied to a regularized objective function. Despite its well-understood theoretical standing and empirical success in supervised learning tasks, SPL has only been applied in a limited way to RL problems, restricting its use to the prioritization of replay data from an experience buffer in deep Q-networks (Ren et al., 2018). Orthogonal to this approach, we will make use of SPL to adaptively select training tasks during learning of the agent.

Furthermore, we will connect the resulting algorithms to the RL-as-inference perspective during the course of this paper. Therefore, we wish to briefly point to several works employing this perspective (Dayan and Hinton, 1997; Toussaint and Storkey, 2006; Deisenroth et al., 2013; Rawlik et al., 2013; Levine, 2018). Taking an inference perspective is beneficial when dealing with inverse problems or problems that require tractable approximations (Hennig et al., 2015; Prince, 2012). Viewing RL as an inference problem naturally motivates regularization methods such as maximum- or relative entropy (Ziebart et al., 2008; Peters et al., 2010; Haarnoja et al., 2018) that have proven highly beneficial in practice. Further, this view allows to rigorously reason about the problem of optimal exploration in RL (Ghavamzadeh et al., 2015). Finally, it stimulates the development of new, and interpretation of existing, algorithms as different approximations to the intractable integrals that need to be computed in probabilistic inference problems (Abdolmaleki et al., 2018; Fellows et al., 2019). This results in a highly principled approach to tackling the challenging problem of RL.

3. Preliminaries

This section introduces the necessary notation for both self-paced and reinforcement learning. Furthermore, the end of Section 3.2 details the intuition of curriculum learning for RL and in particular our approach.

3.1 Self-Paced Learning

The concept of self-paced learning (SPL), as introduced by Kumar et al. (2010) and extended by Jiang et al. (2015), is defined for supervised learning settings, in which a function approximator $y = m(\mathbf{x}, \boldsymbol{\omega})$ with parameters $\boldsymbol{\omega} \in \mathbb{R}^{d_{\boldsymbol{\omega}}}$ is trained w.r.t. a given data set $\mathcal{D} = \left\{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^{d_{\mathbf{x}}},\ y_i \in \mathbb{R},\ i \in [1, N]\right\}$. In this setting, SPL generates a curriculum over the data set $\mathcal{D}$ by introducing a vector $\boldsymbol{\nu} = [\nu_1\ \nu_2\ \dots\ \nu_N] \in [0,1]^N$ of weights ν_i for the entries $(\mathbf{x}_i, y_i)$ in the data set. These weights are automatically adjusted during learning via a 'self-paced regularizer' $f(\alpha, \nu_i)$ in the SPL objective

$$\boldsymbol{\nu}^*, \boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\nu}, \boldsymbol{\omega}}\ r(\boldsymbol{\omega}) + \sum_{i=1}^{N} \left(\nu_i l(\mathbf{x}_i, y_i, \boldsymbol{\omega}) + f(\alpha, \nu_i)\right), \qquad \alpha > 0. \tag{1}$$

The term $r(\boldsymbol{\omega})$ represents potentially employed regularization of the model and $l(\mathbf{x}_i, y_i, \boldsymbol{\omega})$ represents the error in the model prediction $\tilde{y}_i{=}m(\mathbf{x}_i, \boldsymbol{\omega})$ for sample $(\mathbf{x}_i, y_i)$. The motivation for this principle as well as its name are best explained by investigating the solution $\boldsymbol{\nu}^*(\alpha, \boldsymbol{\omega})$ of optimization problem (1) when only optimizing it w.r.t. $\boldsymbol{\nu}$ while keeping α and $\boldsymbol{\omega}$ fixed. Introducing the notation $\nu^*(\alpha, l) = \arg\min_\nu \nu l + f(\alpha, \nu)$, we can define the optimal $\boldsymbol{\nu}$ for given α and $\boldsymbol{\omega}$ as

$$\boldsymbol{\nu}^*(\alpha, \boldsymbol{\omega}) = [\nu^*(\alpha, l(\mathbf{x}_1, y_1, \boldsymbol{\omega}))\ \nu^*(\alpha, l(\mathbf{x}_2, y_2, \boldsymbol{\omega}))\ \dots\ \nu^*(\alpha, l(\mathbf{x}_N, y_N, \boldsymbol{\omega}))].$$

For the self-paced function $f_{\text{Bin}}(\alpha, \nu_i) = -\alpha\nu_i$ initially proposed by Kumar et al. (2010), it holds that

$$\nu^*_{\text{Bin}}(\alpha, l) = \begin{cases} 1, & \text{if } l < \alpha \\ 0, & \text{else.} \end{cases} \tag{2}$$

We see that the optimal weights $\boldsymbol{\nu}^*_{\text{Bin}}(\alpha, \boldsymbol{\omega})$ focus on examples on which the model under the current parameters $\boldsymbol{\omega}$ performs better than a chosen threshold α. By continuously increasing α and updating $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$ in a block-coordinate manner, SPL creates a curriculum consisting of increasingly "hard" training examples w.r.t. the current model. A highly interesting connection between SPL and well-known regularization terms for machine learning has been established by Meng et al. (2017). Based on certain axioms on the self-paced regularizer $f(\alpha, \nu_i)$ (see appendix), Meng et al. (2017) showed that the SPL scheme of alternatingly optimizing (1) w.r.t. $\boldsymbol{\omega}$ and $\boldsymbol{\nu}$ implicitly optimizes the regularized objective

$$\min_{\boldsymbol{\omega}} r(\boldsymbol{\omega}) + \sum_{i=1}^{N} F_\alpha(l(\mathbf{x}_i, y_i, \boldsymbol{\omega})), \qquad F_\alpha(l(\mathbf{x}_i, y_i, \boldsymbol{\omega})) = \int_0^{l(\mathbf{x}_i, y_i, \boldsymbol{\omega})} \nu^*(\alpha, \iota)\, \mathrm{d}\iota. \tag{3}$$

Using the Leibniz integral rule on F_α, we can see that $\nabla_l F_\alpha(l) = \nu^*(\alpha, l)$. Put differently, the weight $\nu^*_i(\alpha, \boldsymbol{\omega})$ encodes how much a decrease in the prediction error $l(\mathbf{x}_i, y_i, \boldsymbol{\omega})$ for the training example $(\mathbf{x}_i, y_i)$ decreases the regularized objective (3). In combination with the previously mentioned axioms on the self-paced regularizer $f(\alpha, \nu_i)$, this allowed Meng et al. (2017) to prove the connection between (1) and (3). Furthermore, they showed that, depending on the chosen self-paced regularizer, the resulting regularizer $F_\alpha(l(\mathbf{x}_i, y_i, \boldsymbol{\omega}))$ corresponds exactly to non-convex regularization terms used in machine learning to e.g.

guide feature selection (Zhang, 2008, 2010). Opposed to feature selection, SPL makes use of these regularizers to attenuate the influence of training examples which the model cannot explain under the current parameters $\boldsymbol{\omega}$. This is done by reducing their contribution to the gradient w.r.t. $\boldsymbol{\omega}$ via the function F_α (see Figure 1). This naturally explains tendencies of SPL to improve learning e.g. in the presence of extreme noise, as empirically demonstrated by Jiang et al. (2015).
To summarize, we have seen that SPL formulates a curriculum over a set of training data as an alternating optimization of weights for the training data $\boldsymbol{\nu}$ given the current model and the model parameters $\boldsymbol{\omega}$ given the current weights. This alternating optimization performs an implicit regularization of the learning objective, suppressing the gradient contribution of samples that the model cannot explain under the current parameters. Empirically, this has been shown to reduce the likelihood of converging to poor local optima. In the subsequent sections, we apply SPL to the problem of reinforcement learning, for which we now introduce the necessary notation.

3.2 Reinforcement Learning

Reinforcement learning (RL) is defined as an optimization problem on a Markov decision process (MDP), a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, p_0 \rangle$ that defines an environment with states $\mathbf{s} \in \mathcal{S}$, actions $\mathbf{a} \in \mathcal{A}$, transition probabilities $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ and initial state distribution $p_0(\mathbf{s})$. Typically $\mathcal{S}$ and $\mathcal{A}$ are discrete spaces or subsets of $\mathbb{R}^n$. RL encompasses approaches that maximize a γ-discounted performance measure

$$\max_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \max_{\boldsymbol{\omega}} \mathbb{E}_{p_0(\mathbf{s}_0), p(\mathbf{s}_{i+1}|\mathbf{s}_i, \mathbf{a}_i), \pi(\mathbf{a}_i|\mathbf{s}_i, \boldsymbol{\omega})} \left[\sum_{i=0}^{\infty} \gamma^i r(\mathbf{s}_i, \mathbf{a}_i) \right] \tag{4}$$

by finding optimal parameters $\boldsymbol{\omega}$ for the policy $\pi(\mathbf{a}|\mathbf{s}, \boldsymbol{\omega})$ through interaction with the environment. A key ingredient to many RL algorithms is the value function

$$V_{\boldsymbol{\omega}}(\mathbf{s}) = \mathbb{E}_{\pi(\mathbf{a}|\mathbf{s}, \boldsymbol{\omega})} \left[r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} \left[V_{\boldsymbol{\omega}}(\mathbf{s}') \right] \right], \tag{5}$$

which encodes the long-term expected discounted reward of following policy $\pi(\cdot|\cdot, \boldsymbol{\omega})$ from state $\mathbf{s}$. The value function (or an estimate of it) is related to the RL objective by $J(\boldsymbol{\omega}) = \mathbb{E}_{p_0(\mathbf{s})} [V_{\boldsymbol{\omega}}(\mathbf{s})]$. In order to exploit learning in multiple MDPs, we need to give the agent ways of generalizing behavior over them. A common approach to accomplish this is to assume a shared state-action space for the MDPs and parameterize the MDP by a contextual parameter $\mathbf{c} \in \mathcal{C} \subseteq \mathbb{R}^m$, i.e. $\mathcal{M}(\mathbf{c}) = \langle \mathcal{S}, \mathcal{A}, p_{\mathbf{c}}, r_{\mathbf{c}}, p_{0,\mathbf{c}} \rangle$. By conditioning the agent's behavior on this context, i.e. $\pi(\mathbf{a}|\mathbf{s}, \mathbf{c}, \boldsymbol{\omega})$, and introducing a distribution $\mu(\mathbf{c})$ over the contextual parameter, we end up with a *contextual* RL objective

$$\max_{\boldsymbol{\omega}} J(\boldsymbol{\omega}, \mu) = \max_{\boldsymbol{\omega}} \mathbb{E}_{\mu(\mathbf{c})} [J(\boldsymbol{\omega}, \mathbf{c})] = \max_{\boldsymbol{\omega}} \mathbb{E}_{\mu(\mathbf{c}), p_{0,\mathbf{c}}(\mathbf{s})} [V_{\boldsymbol{\omega}}(\mathbf{s}, \mathbf{c})]. \tag{6}$$

The value function $V_{\boldsymbol{\omega}}(\mathbf{s}, \mathbf{c})$ now encodes the expected discounted reward of being in states $\mathbf{s}$ *in context* $\mathbf{c}$ and following the conditioned policy $\pi(\mathbf{a}|\mathbf{s}, \mathbf{c}, \boldsymbol{\omega})$, i.e.

$$V_{\boldsymbol{\omega}}(\mathbf{s}, \mathbf{c}) = \mathbb{E}_{\pi(\mathbf{a}|\mathbf{s}, \mathbf{c}, \boldsymbol{\omega})} \left[r_{\mathbf{c}}(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p_{\mathbf{c}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})} \left[V_{\boldsymbol{\omega}}(\mathbf{s}', \mathbf{c}) \right] \right]. \tag{7}$$

This formulation has been investigated by multiple works from different perspectives (Neumann, 2011; Schaul et al., 2015; Modi et al., 2018). Despite the generality of the RL paradigm and its power in formulating the problem of inferring optimal behavior from experience as a stochastic optimization, the practical realization of sophisticated RL algorithms poses many challenges in itself. For example, the extensive function approximations that need to be performed often result in the particular RL algorithm converging to a local optimum of the typically non-convex objectives (4) and (6), which may or may not encode behavior that is able to solve the task (or the distribution of tasks). In the single-task RL objective (4), the only way to avoid such problems is to improve approximations, e.g. by increasing the number of samples, or develop algorithms with strong exploratory behavior. In the contextual case (6), however, there is an appealing different approach.
Assume a task $\mathcal{M}(\mathbf{c})$, in which learning can robustly take place despite aforementioned approximations, e.g. since objective (4) is convex for $\mathcal{M}(\mathbf{c})$ and hence only one optimum exists. Furthermore, consider a second task $\mathcal{M}(\mathbf{c}')$ which now admits multiple optima with different expected rewards. If the solution to $\mathcal{M}(\mathbf{c})$ lies within the basin of attraction of the optimal solution to $\mathcal{M}(\mathbf{c}')$, first learning in $\mathbf{c}$ and afterward in $\mathbf{c}'$ promises to stabilize learning towards the optimal solution for $\mathbf{c}'$. This intuition is at the core of curriculum learning for RL. Looking at (6), a suitable formulation of a curriculum is as a sequence of context distributions $p_i(\mathbf{c})$. This sequence should converge to a desired target distribution $\mu(\mathbf{c})$, i.e. $\lim_{i\to\infty} p_i(\mathbf{c}) = \mu(\mathbf{c})$.
Before we show that self-paced learning induces such a sequence of distributions, we first want to note an important property that is exploited by contextual- and curriculum reinforcement learning (CRL) algorithms especially in continuous domains: A small distance $\|\mathbf{c} - \mathbf{c}'\|$ implies a certain similarity between the tasks $\mathcal{M}(\mathbf{c})$ and $\mathcal{M}(\mathbf{c}')$. Note that the imprecision of this formulation is not by accident but is rather an acknowledgment that the question of similarity between MDPs is a complicated topic on its own. Nonetheless, if in a curriculum, a new training task $\mathbf{c}'$ is generated via additive noise on a task $\mathbf{c}$ in which the agent demonstrates good performance, the property is clearly exploited. Furthermore, policy representations $\pi(\mathbf{a}|\mathbf{s}, \mathbf{c}, \boldsymbol{\omega})$ such as e.g. (deep) neural networks also tend to encode continuity w.r.t. $\mathbf{c}$. We wanted to highlight these observations, as they allow to judge whether a given CRL algorithm, as well as ours, is applicable to a given problem.

4. A Probabilistic Interpretation of Self-Paced Learning

At this point, we have discussed RL and highlighted the problem of policy optimization converging to a local optimum or only converging slowly. Ensuring to learn globally optimal policies with optimal sample complexity in its whole generality is an open problem. However, in Section 3.1 we discussed that for supervised learning, the use of regularizing functions F_α that transform the loss can smooth out local optima created e.g. by noisy training data. Motivated by this insight, we now apply the aforementioned functions to regularize the contextual RL objective, obtaining

$$\max_{\boldsymbol{\omega}} \mathbb{E}_{\mu(\mathbf{c})} \left[F_\alpha(J(\boldsymbol{\omega}, \mathbf{c})) \right]. \tag{8}$$

This objective has two slight differences to objective (3). First, it misses the regularization term $r(\boldsymbol{\omega})$ from (3). Second, objective (8) is defined as an expectation of the regularized

performance $F_\alpha(J(\boldsymbol{\omega}, \mathbf{c}))$ w.r.t. to the context distribution $\mu(\mathbf{c})$ instead of a sum over the regularized performances. This can be seen as a generalization of (3), in which we allow to chose $\mu(\mathbf{c})$ differently from a uniform distribution over a discrete set of values. Regardless of these technical differences, one could readily optimize objective (8) in a supervised learning scenario e.g. via a form of stochastic gradient descent. As argued in Section 3.1, this results in an SPL optimization scheme (1) since the regularizer F_α performs an implicit weighting of the gradients $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}, \mathbf{c})$. In an RL setting, the problem with such a straightforward optimization is that each evaluation of $J(\boldsymbol{\omega}, \mathbf{c})$ and its gradient is typically expensive. If now for given parameters $\boldsymbol{\omega}$ and context $\mathbf{c}$, the regularizer F_α leads to a negligible influence of $J(\boldsymbol{\omega}, \mathbf{c})$ to the gradient of the objective (see Figure 1), evaluating $J(\boldsymbol{\omega}, \mathbf{c})$ wastes the precious resources that the learning agent should carefully utilize. In an RL setting, it is hence crucial to make use of a sampling distribution $p(\mathbf{c})$ that avoids the described wasteful evaluations. At this point, the insight that an SPL weight is equal to the gradient of the regularizing function F_α for the corresponding context, i.e. $\nu^*(\alpha, J(\mathbf{c}, \boldsymbol{\omega})) = \nabla_l F_\alpha(l)|_{l=J(\mathbf{c},\boldsymbol{\omega})}$, directly yields a method for efficiently evaluating objective (8)—that is by sampling a context $\mathbf{c}$ according to its SPL weight $\nu^*(\alpha, J(\mathbf{c}, \boldsymbol{\omega}))$. To make this intuition rigorous, we now introduce a probabilistic view on self-paced learning that views the weights $\boldsymbol{\nu}$ in the SPL objective (1) as probabilities of a distribution over samples. More precisely, we define the categorical probability distribution $p(c{=}i|\boldsymbol{\nu}) = \nu_i$ for $i \in [1, N]$. Note that we restrict ourselves to discrete distributions $p(c{=}i|\boldsymbol{\nu})$ in this section to both ease the exposition and more easily establish connections to the SPL objective introduced in Section 3.1, although the results can be generalized to continuous distributions $\mu(\mathbf{c})$. For $p(c{=}i|\boldsymbol{\nu}) = \nu_i$ to be a valid probability distribution, we only need to introduce the constraint $\sum_{i=1}^N \nu_i = 1$, as $\nu_i \geq 0$ per definition of SPL. Hence, we rewrite the SPL objective (1) as

$$\begin{aligned} \boldsymbol{\nu}^*, \boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\nu}, \boldsymbol{\omega}} \; & r(\boldsymbol{\omega}) + \mathbb{E}_{p(c|\boldsymbol{\nu})}\left[l(\mathbf{x}_c, y_c, \boldsymbol{\omega})\right] + \sum_{i=1}^N f(\alpha, p(c{=}i|\boldsymbol{\nu})), \qquad \alpha > 0 \\ \text{s.t.} \; & \sum_{i=1}^N \nu_i = 1. \end{aligned} \tag{9}$$

Apart from changes in notation, the only difference w.r.t. (1) is the constraint that forces the variables ν_i to sum to 1. Interestingly, depending on the self-paced regularizer $f(\alpha, p(c{=}i|\boldsymbol{\nu}))$, this constraint does not just lead to a normalization of the SPL weights obtained by optimizing objective (1). This is because the previously independent SPL weights $\nu^*(\alpha, l(\mathbf{x}_i, y_i, \boldsymbol{\omega}))$ are now coupled via the introduced normalization constraint. An example of such a "problematic" regularizer is the seminal one $f_{\text{Bin}}(\alpha, \nu_i) = -\alpha\nu_i$ explored by Kumar et al. (2010). With the additional constraint, the optimal solution $\boldsymbol{\nu}^*_{\text{Bin}}$ to (9) simply puts all weight on the sample with the minimum loss instead of sampling uniformly among samples with a loss smaller than α. Although there seems to be no general connection between objective (1) and (9) that holds for arbitrary self-paced regularizers, we can show that for the self-paced regularizer

$$f_{\text{KL},i}(\alpha, \nu_i) = \alpha\nu_i \left(\log(\nu_i) - \log(\mu(c{=}i))\right) - \alpha\nu_i, \tag{10}$$

the value of $\nu^*_{\text{KL},i}(\alpha, \boldsymbol{\omega})$ obtained by optimizing (1) and (9) w.r.t. $\boldsymbol{\nu}$ is identical up to a normalization constant. The user-chosen distribution $\mu(c)$ in (10) represents the likelihood

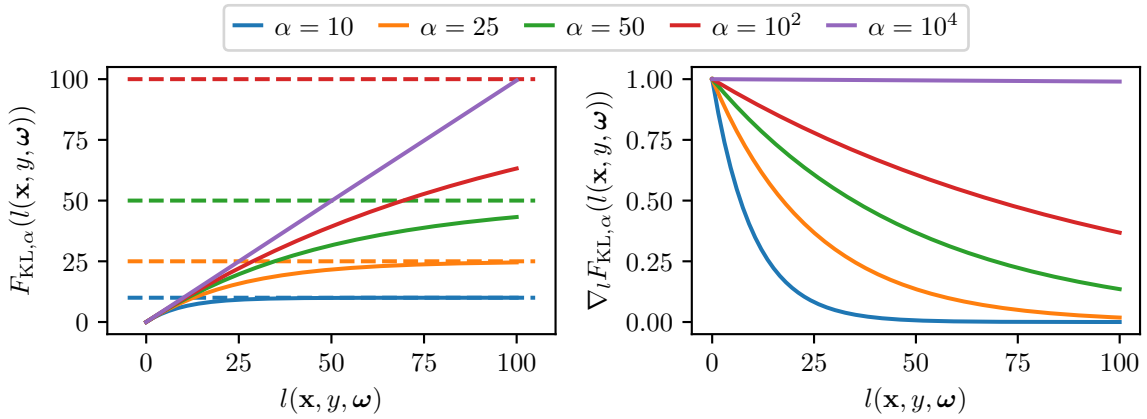


Figure 1: A visualization of the effect of $F_{\mathrm{KL},\alpha}$ (see Equation 11) for different values of α and a single data-point $(\mathbf{x}, y)$. The left plot shows the transformation of the model error $l(\mathbf{x}, y, \boldsymbol{\omega})$ by $F_{\mathrm{KL},\alpha}$. The right plot shows the gradient of $F_{\mathrm{KL},\alpha}$ w.r.t. $l(\mathbf{x}, y, \boldsymbol{\omega})$, i.e. the corresponding weight $\nu^*_{\mathrm{KL}}(\alpha, \boldsymbol{\omega})$.

of $(\mathbf{x}_c, y_c)$ occurring and has the same interpretation as in objective (8). The corresponding function $F_{\mathrm{KL},\alpha,i}$ is given by

$$F_{\mathrm{KL},\alpha,i}(l(\mathbf{x}_i, y_i, \boldsymbol{\omega})) = \int_0^{l(\mathbf{x}_i, y_i, \boldsymbol{\omega})} \nu^*_{\mathrm{KL},i}(\alpha, \iota)\, \mathrm{d}\iota = \mu(c{=}i)\alpha \left(1 - \exp\left(-\frac{1}{a} l(\mathbf{x}_i, y_i, \boldsymbol{\omega})\right)\right) \quad (11)$$

and is visualized in Figure 1. Note the additional subscript i in both $f_{\mathrm{KL},i}$ and $F_{\mathrm{KL},\alpha,i}$. This extra subscript arises due to the appearance of the likelihood term $\mu(c{=}i)$ in both formulas, resulting in an individual regularizer for each sample $(\mathbf{x}_i, y_i)$. As can be seen, $F_{\mathrm{KL},\alpha,i}(l)$ exhibits a squashing effect to limit the attained loss l to a maximum value of α. The closer the non-regularized loss l attains this maximum value of α, the more it is treated as a constant value by $F_{\mathrm{KL},\alpha,i}(l)$. For l increasingly smaller than α, a change in the non-regularized loss l leads to an increasingly linear change in the regularized loss $F_{\mathrm{KL},\alpha,i}(l)$. More interestingly, using $f_{\mathrm{KL},i}(\alpha, \nu_i)$ in objective (9) results in a KL-Divergence penalty to $\mu(c)$. Theorem 1 summarizes these findings. The proof can be found in the appendix.

Theorem 1 *Alternatingly solving*

$$\min_{\boldsymbol{\omega}, \boldsymbol{\nu}} \mathbb{E}_{p(c|\boldsymbol{\nu})} \left[l(\mathbf{x}_c, y_c, \boldsymbol{\omega}) \right] + \alpha D_{KL} \left(p(c|\boldsymbol{\nu}) \parallel \mu(c) \right)$$

w.r.t. $\boldsymbol{\omega}$ *and* $\boldsymbol{\nu}$ *is a majorize-minimize scheme applied to the regularized objective*

$$\min_{\boldsymbol{\omega}} \mathbb{E}_{\mu(c)} \left[\alpha \left(1 - \exp\left(-\frac{1}{\alpha} l(\mathbf{x}_c, y_c, \boldsymbol{\omega}) \right) \right) \right].$$

In the following section, we make use of the insights summarized in Theorem 1 to motivate a curriculum as an effective evaluation of the regularized RL objective (8) under the particular choice $F_\alpha = F_{\mathrm{KL},\alpha,i}$.

5. Self-Paced Learning for Reinforcement Learning

Obtaining an efficient way of optimizing objective (8) with $F_\alpha = F_{\mathrm{KL},\alpha,i}$ is as easy as exploiting Theorem 1 to define the alternative objective

$$\max_{\boldsymbol{\omega},\boldsymbol{\nu}} \mathbb{E}_{p(\mathbf{c}|\boldsymbol{\nu})}\left[J(\boldsymbol{\omega},\mathbf{c})\right] - \alpha D_{\mathrm{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}) \parallel \mu(\mathbf{c})\right).$$

As discussed in the previous section, this formulation introduces a way of computing the desired sampling distribution that efficiently evaluates objective (8) given the current agent parameters $\boldsymbol{\omega}$ by optimizing the above optimization problem w.r.t. $\boldsymbol{\nu}$. As discussed in Section 3.1, $p(\mathbf{c}|\boldsymbol{\nu})$ will assign probability mass to a context $\mathbf{c}$ based on its contribution to the gradient of objective (8). Before we look at the application to RL problems, we will introduce a regularization that is an important ingredient to achieve practicality. More precisely, we introduce a KL divergence constraint between subsequent context distributions $p(\mathbf{c}|\boldsymbol{\nu})$ and $p(\mathbf{c}|\boldsymbol{\nu}')$, yielding

$$\begin{aligned}\max_{\boldsymbol{\omega},\boldsymbol{\nu}}\ & \mathbb{E}_{p(\mathbf{c}|\nu)}\left[J(\boldsymbol{\omega},\mathbf{c})\right] - \alpha D_{\mathrm{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}) \parallel \mu(\mathbf{c})\right)\\ \text{s.t. }& D_{\mathrm{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}) \,\middle\|\, p(\mathbf{c}|\boldsymbol{\nu}')\right) \leq \epsilon,\end{aligned} \tag{12}$$

with $\boldsymbol{\nu}'$ being the parameters of the previously computed context distribution. In a practical algorithm, this secondary regularization is important because the expected performance $J(\boldsymbol{\omega},\mathbf{c})$ is approximated by a learned value function, which may not predict accurate values for contexts not likely under $p(\mathbf{c}|\boldsymbol{\nu}')$. The KL divergence constraint helps to avoid exploiting these false estimates too greedily. Furthermore, it forces the distribution over contextual variables, and hence tasks, to gradually change, which we argued to be beneficial for RL algorithms at the end of Section 3.2. From a theoretical perspective on SPL, the constraint changes the form of $\boldsymbol{\nu}^*$ making it not only dependent on α and $\boldsymbol{\omega}$, but also on the previous parameter $\boldsymbol{\nu}'$. Although it may be possible to relate this modification to a novel regularizer $F_{\alpha,i}$, we do not pursue this idea here but rather connect (12) to the RL-as-inference perspective in Section 9, where we can show highly interesting similarities to the well-known concept of tempering in inference. To facilitate the intuition of the proposed curriculum and its usage, we, however, first present applications and evaluations in the following sections.
An important design decision for such applications is the schedule for α, i.e. the parameter of the regularizing function F_α. As can be seen in (12), α corresponds to the trade-off between reward maximization and progression to $\mu(\mathbf{c})$. In a supervised learning scenario, it is preferable to increase α as slowly as possible to gradually transform the objective from an easy version towards the target one. In an RL setting, each algorithm iteration requires the collection of data from the (real) system. Since the required amount of system interaction should be minimized, we cannot simply choose very small step sizes for α, as this would lead to a slower than necessary progression towards $\mu(\mathbf{c})$. In the implementations in sections 6 and 7, the parameter α is chosen such that the KL divergence penalty w.r.t. the current context distribution $p(\mathbf{c}|\boldsymbol{\nu}_k)$ is in constant proportion ζ to the expected reward under this current context distribution and current policy parameters $\boldsymbol{\omega}_k$

$$\alpha_k = \mathcal{B}(\boldsymbol{\nu}_k,\boldsymbol{\omega}_k) = \zeta \frac{\mathbb{E}_{p(\mathbf{c}|\boldsymbol{\nu}_k)}\left[J(\boldsymbol{\omega}_k,\mathbf{c})\right]}{D_{\mathrm{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}_k) \parallel \mu(\mathbf{c})\right)}. \tag{13}$$

For the first K_α iterations, we set α to zero, i.e. only focus on maximizing the reward under $p(\mathbf{c}|\boldsymbol{\nu})$. In combination with an initial context distribution $p(\mathbf{c}|\boldsymbol{\nu}_0)$ covering large parts of the context space, this allows to tailor the context distribution to the learner in the first iterations by focusing on tasks in which it performs best under the initial parameters. Note that this schedule is a naive choice, that nonetheless worked sufficiently well in our experiments. In Section 8, we revisit this design choice and investigate it more carefully.

6. Application to Episodic Reinforcement Learning

In this section, we implement and evaluate our formulation of SPL for RL in a slightly different way than the "full" RL setting, which has been described in Section 3.2 and will be evaluated in the next section. Instead, we frame RL as a black-box optimization problem (Conn et al., 2009; Hansen et al., 2010). This setting is interesting for two reasons: Firstly, it has been and still is a core approach to perform RL on real (robotic) systems (Kupcsik et al., 2013; Parisi et al., 2015; Ploeger et al., 2020), where "low-level" policies such as Dynamic- and Probabilistic Movement Primitives (Schaal, 2006; Paraschos et al., 2013) or PD-control laws (Berkenkamp et al., 2016) are commonly used to ensure smooth and stable trajectories while keeping the dimensionality of the search space reasonably small. Secondly, the different mechanics of the employed episodic RL algorithm and the resulting different implementation of objective (12) serve as another validation of our SPL approach to CRL apart from the deep RL experiments in the next section. Readers not interested in or familiar with the topic of black-box optimization (and episodic RL) can skip this section and continue to the experiments with deep RL algorithms in Section 7.

The episodic RL setting arises if we introduce an additional "low-level" policy $\pi(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^{d_{\boldsymbol{\theta}}}$ and change the policy introduced in Section 3.2 to not generate actions given the current state and context, but only generate a parameter $\boldsymbol{\theta}$ for the low-level policy given the current context, i.e. $\pi(\boldsymbol{\theta}|\mathbf{c}, \boldsymbol{\omega})$. Defining the expected reward for a parameter $\boldsymbol{\theta}$ in context $\mathbf{c}$ as

$$r(\boldsymbol{\theta}, \mathbf{c}) = \mathbb{E}_{p_{0,\mathbf{c}}(\mathbf{s})}\left[V_{\pi(\mathbf{a}|\mathbf{s},\boldsymbol{\theta})}(\mathbf{s}, \mathbf{c})\right], \tag{14}$$

we see that we can simply interpret $r(\boldsymbol{\theta}, \mathbf{c})$ as a function that, due to its complicated nature, does only allow for noisy observations of its function value without any gradient information. The noise in function observations arises from the fact that a rollout of policy $\pi(\mathbf{a}|\mathbf{s}, \boldsymbol{\theta})$ in a context $\mathbf{c}$ corresponds to approximating the expectations in $r(\boldsymbol{\theta}, \mathbf{c})$ with a single sample. As a black-box optimizer for the experiments, we choose the contextual relative entropy policy search (C-REPS) algorithm (Neumann, 2011; Kupcsik et al., 2013; Parisi et al., 2015), which frames the maximization of (14) over a task distribution $\mu(\mathbf{c})$ as a repeated entropy-regularized optimization

$$\max_{q(\boldsymbol{\theta},\mathbf{c})} \mathbb{E}_{q(\boldsymbol{\theta},\mathbf{c})}\left[r(\boldsymbol{\theta}, \mathbf{c})\right] \quad \text{s.t. } D_{\mathrm{KL}}\left(q(\boldsymbol{\theta}, \mathbf{c}) \parallel p(\boldsymbol{\theta}, \mathbf{c})\right) \leq \epsilon \quad \int q(\boldsymbol{\theta}, \mathbf{c})\, \mathrm{d}\boldsymbol{\theta} = \mu(\mathbf{c}) \;\forall \mathbf{c} \in \mathcal{C},$$

where $p(\boldsymbol{\theta}, \mathbf{c}) = p(\boldsymbol{\theta}|\mathbf{c})\mu(\mathbf{c})$ is the distribution obtained in the previous iteration. Note that the constraint in the above optimization problem implies that only the policy $q(\boldsymbol{\theta}|\mathbf{c})$ is optimized since the constraint requires that $q(\boldsymbol{\theta}, \mathbf{c}) = q(\boldsymbol{\theta}|\mathbf{c})\mu(\mathbf{c})$. This notation is common for this algorithm as it eases the derivations of the solution via the concept of Lagrangian multipliers. Furthermore, this particular form of the C-REPS algorithm allows for

Algorithm 1 Self-Paced Episodic Reinforcement Learning (SPRL)

Input: Initial context distribution- and policy parameters $\boldsymbol{\nu}_0$ and $\boldsymbol{\omega}_0$, Target context distribution $\mu(\mathbf{c})$, KL penalty proportion ζ, Offset K_α, Number of iterations K, Rollouts per policy update M, Relative entropy bound ϵ
for $k = 1$ **to** K **do**
  Collect Data:
  Sample contexts: $\mathbf{c}_i \sim p(\mathbf{c}|\boldsymbol{\nu}_{k-1}),\ i \in [1, M]$
  Sample parameters: $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}|\mathbf{c}_i, \boldsymbol{\omega}_{k-1})$
  Execute $\pi(\cdot|\mathbf{s}, \boldsymbol{\theta}_i)$ in $\mathbf{c}_i$ and observe reward: $r_i = r(\boldsymbol{\theta}_i, \mathbf{c}_i)$
  Create sample set: $\mathcal{D}_k = \{(\boldsymbol{\theta}_i, \mathbf{c}_i, r_i)|i \in [1, M]\}$
  Update Policy and Context Distributions:
  Update schedule: $\alpha_k = 0,$ if $k \leq K_\alpha,$ else $\mathcal{B}(\boldsymbol{\nu}_{k-1}, \boldsymbol{\omega}_{k-1})$ (13)
  Optimize dual function: $\left[\eta_q^*, \eta_{\tilde{q}}^*, V^*\right] \leftarrow \arg\min \mathcal{G}(\eta_q, \eta_{\tilde{q}}, V)$ (18)
  Calculate sample weights: $[w_i, \tilde{w}_i] \leftarrow \left[\exp\left(\frac{A(\boldsymbol{\theta}_i, \mathbf{c}_i)}{\eta_q^*}\right), \exp\left(\frac{\beta(\mathbf{c}_i)}{\alpha_k + \eta_{\tilde{q}}^*}\right)\right]$ (16), (17)
  Infer new parameters: $[\boldsymbol{\omega}_k, \boldsymbol{\nu}_k] \leftarrow \{(w_i, \tilde{w}_i, \boldsymbol{\theta}_i, \mathbf{c}_i)|i \in [1, M]\}$
end for

a straightforward incorporation of SPL, simply replacing the constraint $\int q(\boldsymbol{\theta}, \mathbf{c}) d\boldsymbol{\theta} = \mu(\mathbf{c})$ by a penalty term on the KL divergence between $q(\mathbf{c}) = \int q(\boldsymbol{\theta}, \mathbf{c}) d\boldsymbol{\theta}$ and $\mu(\mathbf{c})$

$$\begin{aligned} &\max_{q(\boldsymbol{\theta},\mathbf{c})} \mathbb{E}_{q(\boldsymbol{\theta},\mathbf{c})}\left[r(\boldsymbol{\theta}, \mathbf{c})\right] - \alpha D_{\mathrm{KL}}\left(q(\mathbf{c}) \parallel \mu(\mathbf{c})\right) \\ &\quad \text{s.t. } D_{\mathrm{KL}}\left(q(\boldsymbol{\theta}, \mathbf{c}) \parallel p(\boldsymbol{\theta}, \mathbf{c})\right) \leq \epsilon. \end{aligned} \tag{15}$$

The above objective does not yet include the parameters $\boldsymbol{\omega}$ or $\boldsymbol{\nu}$ of the policy or the context distribution to be optimized. This is because both C-REPS and also our implementation of SPL for episodic RL solve the above optimization problem analytically to obtain a re-weighting scheme for samples $(\boldsymbol{\theta}_i, \mathbf{c}_i) \sim p(\boldsymbol{\theta}|\mathbf{c}, \boldsymbol{\omega}_k) p(\mathbf{c}|\boldsymbol{\nu}_k)$ based on the observed rewards $r(\boldsymbol{\theta}_i, \mathbf{c}_i)$. The next parameters $\boldsymbol{\omega}_{k+1}$ and $\boldsymbol{\nu}_{k+1}$ are then found by a maximum-likelihood fit to the set of weighted samples. The following section will detail some of the practical considerations necessary for this.

6.1 Algorithmic Implementation

Solving (15) analytically using the technique of Lagrangian multipliers, we obtain the following form for the variational distributions

$$q(\boldsymbol{\theta}, \mathbf{c}) \propto p(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\omega}_k, \boldsymbol{\nu}_k) \exp\left(\frac{r(\boldsymbol{\theta}, \mathbf{c}) - V(\mathbf{c})}{\eta_q}\right) = p(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\omega}_k, \boldsymbol{\nu}_k) \exp\left(\frac{A(\boldsymbol{\theta}, \mathbf{c})}{\eta_q}\right), \tag{16}$$

$$q(\mathbf{c}) \propto p(\mathbf{c}|\boldsymbol{\nu}_k) \exp\left(\frac{V(\mathbf{c}) + \alpha(\log(\mu(\mathbf{c})) - \log(p(\mathbf{c}|\boldsymbol{\nu}_k)))}{\alpha + \eta_{\tilde{q}}}\right) = p(\mathbf{c}|\boldsymbol{\nu}_k) \exp\left(\frac{\beta(\mathbf{c})}{\alpha + \eta_{\tilde{q}}}\right), \tag{17}$$

with η_q, $\eta_{\tilde{q}}$ as well as $V(\mathbf{c})$ being Lagrangian multipliers that are found by solving the dual objective

$$\mathcal{G} = (\eta_q + \eta_{\tilde{q}})\epsilon + \eta_q \log\left(\mathbb{E}_p\left[\exp\left(\frac{A(\boldsymbol{\theta}, \mathbf{c})}{\eta_q}\right)\right]\right) + (\alpha + \eta_{\tilde{q}}) \log\left(\mathbb{E}_p\left[\exp\left(\frac{\beta(\mathbf{c})}{\alpha + \eta_{\tilde{q}}}\right)\right]\right). \tag{18}$$

The derivation of the dual objective, as well as the solution to objective (15), are shown in the appendix. As previously mentioned, in practice the algorithm has only access to a set of samples $\mathcal{D} = \{(\boldsymbol{\theta}_i, \mathbf{c}_i, r_i) | i \in [1, M]\}$ and hence the analytic solutions (16) and (17) are approximated by re-weighting the samples via weights w_i. To compute the optimal weights w_i, the multipliers V^*, η_q^* and $\eta_{\tilde{q}}^*$ need to be obtained by minimizing the dual (18), to which two approximations are introduced: First, the expectations w.r.t. $p(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\omega}, \boldsymbol{\nu})$ (abbreviated as p in Equation 18) are replaced by a sample-estimate from the collected samples in $\mathcal{D}$. Second, we introduce a parametric form for the value function $V(\mathbf{c}) = \boldsymbol{\chi}^T \phi(\mathbf{c})$ with a user-chosen feature function $\phi(\mathbf{c})$, such that we can optimize (18) w.r.t. $\boldsymbol{\chi}$ instead of V.
After finding the minimizers $\boldsymbol{\chi}^*$, η_q^* and $\eta_{\tilde{q}}^*$ of (18), the weights w_i are then given by the exponential terms in (16) and (17). The new policy- and context distribution parameters are fitted via maximum likelihood to the set of weighted samples. In our implementation, we use Gaussian context distributions and policies. To account for the error that originates from the sample-based approximation of the expectations in (18), we enforce the KL divergence constraint $D_{\mathrm{KL}}\left(p(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\omega}_k, \boldsymbol{\nu}_k) \parallel q(\boldsymbol{\theta}, \mathbf{c}|\boldsymbol{\omega}_{k+1}, \boldsymbol{\nu}_{k+1})\right) \leq \epsilon$ when updating the policy and context distribution. Again, details on this maximum likelihood step can be found in the appendix. To compute the schedule for α according to (13), we approximate the expected reward under the current policy with the mean of the observed rewards, i.e. $\mathbb{E}_{p(\mathbf{c}|\boldsymbol{\nu}_k)}\left[J(\boldsymbol{\omega}_k, \mathbf{c})\right] \approx \frac{1}{M}\sum_{i=1}^{M} r_i$. The overall procedure is summarized in Algorithm 1.

6.2 Experiments

We now evaluate the benefit of the SPL paradigm in the episodic RL scenario (SPRL). Besides facilitating learning on a diverse set of tasks, we are also interested in the idea of facilitating the learning of a hard target task via a curriculum. This modulation can be achieved by choosing $\mu(\mathbf{c})$ to be a narrow probability distribution focusing nearly all probability density on the particular target task. To judge the benefit of our SPL adaptation for these endeavors, we compared our implementation to C-REPS, CMA-ES (Hansen et al., 2003), GoalGAN (Florensa et al., 2018) and SAGG-RIAC (Baranes and Oudeyer, 2010). With CMA-ES being a non-contextual algorithm, we only use it in experiments with narrow target distributions, where we then train and evaluate only on the mean of the target context distributions. We will start with a simple point-mass problem, where we evaluate the benefit of our algorithm for broad and narrow target distributions. We then turn towards more challenging tasks, such as a modified version of the reaching task implemented in the OpenAI Gym simulation environment (Brockman et al., 2016) and a sparse ball-in-a-cup task. Given that GoalGAN and SAGG-RIAC are algorithm agnostic curriculum generation approaches, we combine them with C-REPS to make the results as comparable as possible.
In all experiments, we use radial basis function (RBF) features to approximate the value function $V(\mathbf{c})$, while the policy $p(\boldsymbol{\theta}|\mathbf{c}, \boldsymbol{\omega}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{A}_{\boldsymbol{\omega}}\phi(\mathbf{c}), \boldsymbol{\Sigma}_{\boldsymbol{\omega}})$ uses linear features $\phi(\mathbf{c})$.

SPRL and C-REPS always use the same number of RBF features for a given environment. SPRL always starts with a wide initial sampling distribution $p(\mathbf{c}|\boldsymbol{\nu}_0)$ that, in combination with setting $\alpha = 0$ for the first K_α iterations, allows the algorithm to automatically choose the initial tasks on which learning should take place. After the first K_α iterations, we then choose α following the scheme outlined in the previous section. Experimental details that are not mentioned here to keep the section short can be found in the appendix. [1]

6.2.1 Point-Mass Environment

In the first environment, the agent needs to steer a point-mass in a two-dimensional space from the starting position $[0 \;\; 5]$ to the goal position at the origin. The dynamics of the point-mass are described by a simple linear system subject to a small amount of Gaussian noise. Complexity is introduced by a wall at height $y = 2.5$, which can only be traversed through a gate. The x-position and width of the gate together define a task $\mathbf{c}$. If the point-mass crashes into the wall, the experiment is stopped and the reward is computed based on the current position. The reward function is the exponentiated negative distance to the goal position with additional L2-Regularization on the generated actions. The point-mass is controlled by two linear controllers, whose parameters need to be tuned by the agent. The controllers are switched as soon as the point-mass reaches the height of the gate, which is why the desired y-position of the controllers are fixed to 2.5 (the height of

1. Code is publicly available under `https://github.com/psclklnk/self-paced-rl`.

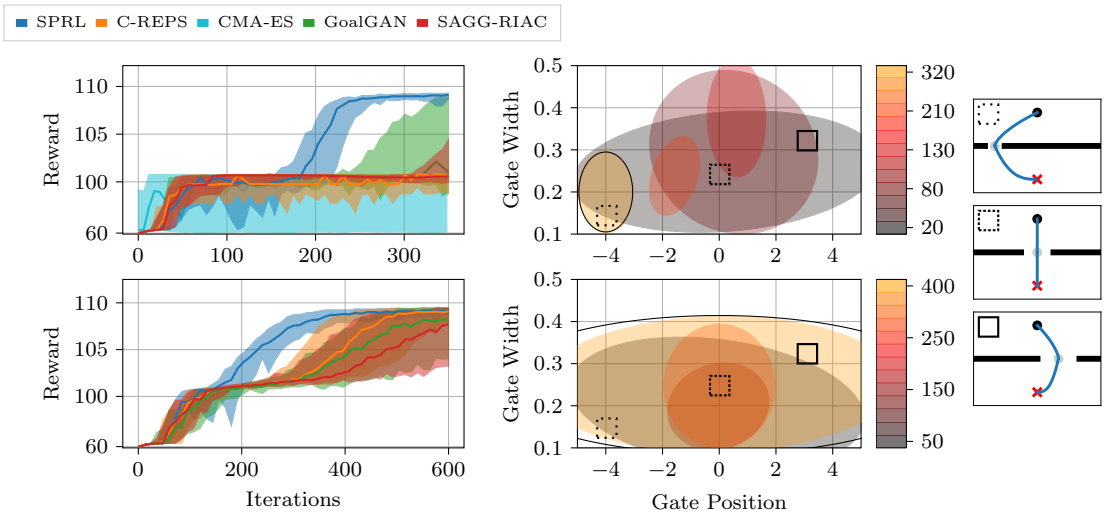


Figure 2: Left: Reward in the "precision" (top row) and "global" setting (bottom row) on the target context distributions in the gate environment. Thick lines represent the 50%-quantiles and shaded areas the intervals from 10%- to 90%-quantile of 40 algorithm executions. Middle: Evolution of the sampling distribution $p(\mathbf{c}|\boldsymbol{\nu})$ (colored areas) of one SPRL run together with the target distribution $\mu(\mathbf{c})$ (black line). Right: Task visualizations for different gate positions and widths. The boxes mark the corresponding positions in the context space.

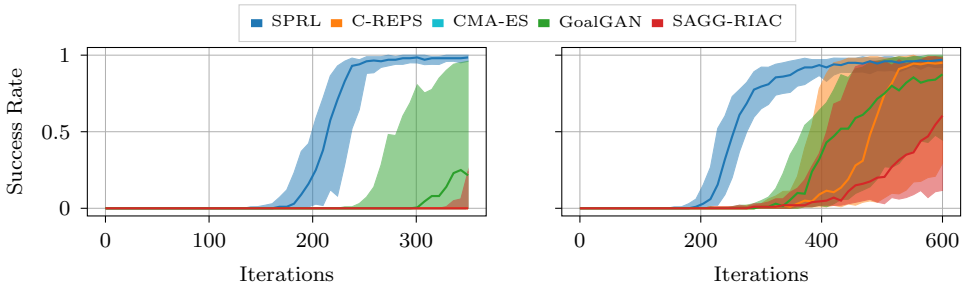


Figure 3: Success rates in the "precision" (left) and "global" setting (right) of the gate environment. Thick lines represent the 50%-quantiles and shaded areas show the intervals from 10%- to 90%-quantile. Quantiles are computed using 40 algorithm executions.

the gate) and 0, while all other parameters are controlled by the policy π, making $\boldsymbol{\theta}$ a 14-dimensional vector. We evaluate two setups in this gate environment, which differ in their target context distribution $\mu(\mathbf{c})$: In the first one, the agent needs to be able to steer through a very narrow gate far from the origin ("precision") and in the second it is required to steer through gates with a variety of positions and widths ("global"). The two target context distributions are shown in Figure 2. Figure 2 further visualizes the obtained rewards for the investigated algorithms, the evolution of the sampling distribution $p(\mathbf{c}|\boldsymbol{\nu})$ as well as tasks from the environment. In the "global" setting, we can see that SPRL converges significantly faster to the optimum than the other algorithms while in the "precision" setting, SPRL avoids a local optimum to which C-REPS and CMA-ES converge and which, as can be seen in Figure 3, does not encode desirable behavior. Furthermore, both curriculum learning algorithms SAGG-RIAC and GoalGAN only slowly escape this local optimum in the "precision" setting. We hypothesize that this slow convergence to the optimum is caused by SAGG-RIAC and GoalGAN not having a notion of a target distribution. Hence, these algorithms cannot guide the sampling of contexts to sample relevant tasks according to $\mu(\mathbf{c})$. This is especially problematic if $\mu(\mathbf{c})$ covers only a small fraction of the context space with a non-negligible probability density. The visualized sampling distributions in Figure 2 indicate that tasks with wide gates positioned at the origin seem to be easier to solve starting from the initially zero-mean Gaussian policy, as in both settings SPRL first focuses on these kinds of tasks and then subsequently changes the sampling distributions to match the target distribution. Interestingly, the search distribution of CMA-ES did not always converge in the "precision" setting, as can be seen in Figure 2. This behavior persisted across various hyperparameters and population sizes.

6.2.2 Reacher Environment

For the next evaluation, we modify the three-dimensional reacher environment of the OpenAI Gym toolkit. In our version, the goal is to move the end-effector along the surface of a table towards the goal position while avoiding obstacles that are placed on the table. With the obstacles becoming larger, the robot needs to introduce a more pronounced curve movement to reach the goal without collisions. To simplify the visualization of the task distribution,

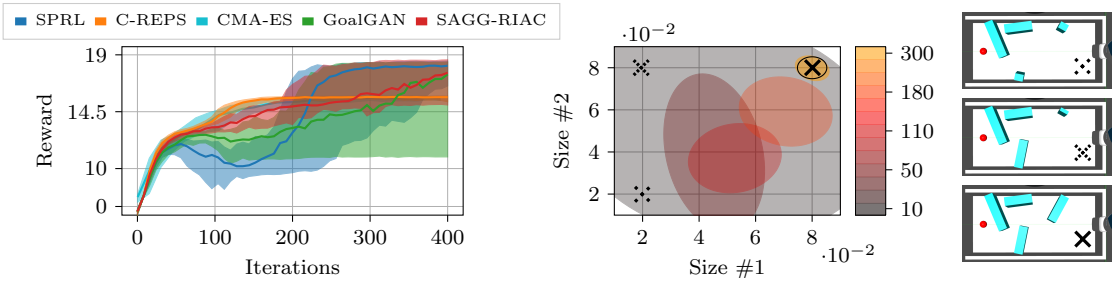


Figure 4: Left: 50%-quantiles (thick lines) and intervals from 10%- to 90%-quantile (shaded areas) of the reward in the reacher environment. Quantiles are computed over 40 algorithm runs. Middle: The sampling distribution $p(\mathbf{c}|\boldsymbol{\nu})$ at different iterations (colored areas) of one SPRL run together with the target distribution (black line). Right: Task visualizations for different contexts with black crosses marking the corresponding positions in context space.

we only allow two of the four obstacles to vary in size. The sizes of those two obstacles make up a task $\mathbf{c}$ in this environment. Just as in the first environment, the robot should not crash into the obstacles, and hence the movement is stopped if one of the four obstacles is touched. The policy π encodes a ProMP (Paraschos et al., 2013), from which movements are sampled during training. In this task, $\boldsymbol{\theta}$ is a 40-dimensional vector.
Looking at Figure 4, we can see that C-REPS and CMA-ES find a worse optimum compared to SPRL. This local optimum does—just as in the previous experiment—not encode optimal behavior, as we can see in Figure 5. GoalGAN and SAGG-RIAC tend to find the same optimum as SPRL, however with slower convergence. This is nonetheless surprising given that—just as for the "precision" setting of the previous experiment—the algorithm deals with a narrow target context distribution. Although the 10%-90% quantile of SAGG-RIAC and GoalGAN contain policies that do not manage to solve the task (i.e. are below the performance of C-REPS), the performance is in stark contrast to the performance in

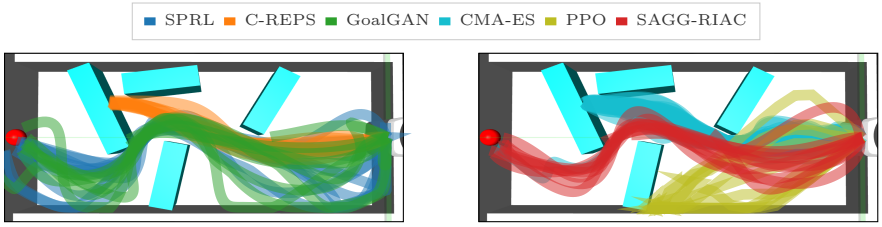


Figure 5: Trajectories generated by final policies learned with different algorithms in the reacher environment. The trajectories should reach the red dot while avoiding the cyan boxes. Please note that the visualization is not completely accurate, as we did not account for the viewpoint of the simulation camera when plotting the trajectories.

the previously discussed "precision" setting, in which the majority of runs did not solve the task. Nonetheless, the 10%-50% quantile of the performance displayed in Figure 4 still indicates the expected effect that SPRL leverages the knowledge of the target distribution to yield faster convergence to the optimal policy in the median case.
Another interesting artifact is the initial decrease in performance of SPRL between iterations $50 - 200$. This can be accounted to the fact that in this phase, the intermediate distribution $p(\mathbf{c}|\boldsymbol{\nu})$ only assigns negligible probability density on areas covered by $\mu(\mathbf{c})$ (see Figure 4). Hence the agent performance on $\mu(\mathbf{c})$ during this stage is completely dependent on the extrapolation behavior of the agent, which seems to be rather poor in this setting. This once more illustrates the importance of appropriate transfer of behavior between tasks, which is, however, out of the scope of this paper.
The sampling distributions visualized in Figure 4 indicate that SPRL focuses on easier tasks with smaller obstacle sizes first and then moves on to the harder, desired tasks. Figure 5 also shows that PPO (Schulman et al., 2017), a step-based reinforcement learning algorithm, is not able to solve the task after the same amount of interaction with the environment, emphasizing the complexity of the learning task.

6.2.3 Sparse Ball-in-a-Cup

We conclude this experimental evaluation with a ball-in-a-cup task, in which the reward function exhibits a significant amount of sparsity by only returning a reward of 1 minus an L2 regularization term on the policy parameters, if the ball is in the cup after the policy execution, and 0 otherwise. The robotic platform is a Barrett WAM, which we simulate using the MuJoCo physics engine (Todorov et al., 2012). The policy represents again a ProMP encoding the desired position of the first, third and fifth joint of the robot. Achieving the desired task with a poor initial policy is an unlikely event, leading to mostly uninformative rewards and hence poor learning progress. However, as can be seen in Figure 6, giving the learning agent control over the diameter of the cup significantly improves the learning

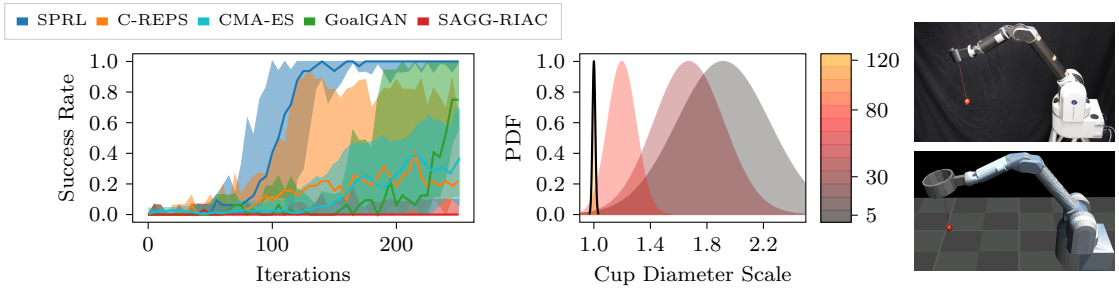


Figure 6: Left: 50%-quantiles (thick lines) and intervals from 10%- to 90%-quantile (shaded areas) of the success rates for the sparse ball-in-a-cup task. Quantiles are computed from the 10 best runs out of 20. Middle: The sampling distribution $p(c|\boldsymbol{\nu})$ at different iterations (colored areas) of one SPRL run together with the target distribution $\mu(c)$ (black line). Right: Task visualization on the real robot (upper) and in simulation with a scale of 2.5 (lower).

Algorithm 2 Self-Paced Deep Reinforcement Learning (SPDL)

Input: Initial context distribution- and policy parameters $\boldsymbol{\nu}_0$ and $\boldsymbol{\omega}_0$, Target context distribution $\mu(\mathbf{c})$, KL penalty proportion ζ and offset K_α, Number of iterations K, Rollouts per policy update M, Relative entropy bound ϵ
for $k = 1$ **to** K **do**
  Agent Improvement:
  Sample contexts: $\mathbf{c}_i \sim p(\mathbf{c}|\boldsymbol{\nu}_k),\ i \in [1, M]$
  Rollout trajectories: $\boldsymbol{\tau}_i \sim p(\boldsymbol{\tau}|\mathbf{c}_i, \boldsymbol{\omega}_k),\ i \in [1, M]$
  Obtain $\boldsymbol{\omega}_{k+1}$ from RL algorithm of choice using $\mathcal{D}_k = \{(\mathbf{c}_i, \boldsymbol{\tau}_i)|i \in [1, M]\}$
  Estimate $\tilde{V}_{\boldsymbol{\omega}_{k+1}}(\boldsymbol{s}_{i,0}, \mathbf{c}_i)$ (or use estimate of RL agent) for contexts $\mathbf{c}_i$
  Context Distribution Update:
  IF $k \leq K_\alpha$: Obtain $\boldsymbol{\nu}_{k+1}$ from (19) with $\alpha_k = 0$
  ELSE: Obtain $\boldsymbol{\nu}_{k+1}$ optimizing (19), using $\alpha_k = \mathcal{B}(\boldsymbol{\nu}_k, \mathcal{D}_k)$ (13)
end for

progress by first training with larger cups and only progressively increasing the precision of the movement to work with smaller cups. Having access to only 16 samples per iteration, the algorithms did not always learn to achieve the task. However, the final policies learned by SPRL outperform the ones learned by C-REPS, CMA-ES, GoalGAN and SAGG-RIAC. The movements learned in simulation were finally applied to the robot with a small amount of fine-tuning.

7. Application to Step-Based Reinforcement Learning

The experiments in the previous section demonstrate that the self-paced learning paradigm can indeed be beneficial in the episodic RL —or black-box optimization— setting, so that as a next step we want to investigate its application when using a stochastic policy of the form $\pi(\mathbf{a}|\mathbf{s}, \mathbf{c}, \boldsymbol{\omega})$. In this setting, we derive an implementation of SPL that is agnostic to the RL algorithm of choice by using the possibility of updating the SPL objective (12) in a block-coordinate manner w.r.t. $\boldsymbol{\omega}$ and $\boldsymbol{\nu}$. The resulting approximate implementation allows to create learning agents following the SPL paradigm using arbitrary RL algorithms by making use of the value functions that the RL algorithms estimate during policy optimization.

7.1 Algorithmic Implementation

When optimizing (12) w.r.t. the policy parameters $\boldsymbol{\omega}$ under the current context distribution $p(\mathbf{c}|\boldsymbol{\nu}_k)$ using an RL algorithm of choice, a data set $\mathcal{D}_k$ of trajectories is generated

$$\mathcal{D}_k = \{(\mathbf{c}_i, \boldsymbol{\tau}_i) \mid \mathbf{c}_i \sim p(\mathbf{c}|\boldsymbol{\nu}_k), \boldsymbol{\tau}_i \sim p(\boldsymbol{\tau}|\mathbf{c}_i, \boldsymbol{\omega}_k), i \in [1, M]\}\,.$$
$$\boldsymbol{\tau}_i = \{(\mathbf{s}_{i,j}, \mathbf{a}_{i,j}, r_{i,j})|\mathbf{a}_{i,j} \sim p(\mathbf{a}|\mathbf{s}_{i,j}, \mathbf{c}_i, \boldsymbol{\omega}_k), \mathbf{s}_{i,j+1} \sim p_{\mathbf{c}_i}(\mathbf{s}|\mathbf{s}_{i,j}, \mathbf{a}_{i,j}), \mathbf{s}_{i,0} \sim p_{0,\mathbf{c}_i}(\mathbf{s}), j{=}1, \ldots\}.$$

One unifying property of many RL algorithms is their reliance on estimating the state-value function $V_{\boldsymbol{\omega}}(\mathbf{s}_0, \mathbf{c})$, each in their respective way, as a proxy to optimizing the policy. We make use of this approximated value function $\tilde{V}_{\boldsymbol{\omega}}(\mathbf{s}_0, \mathbf{c})$ (note the $\sim$ indicating the approximation) to compute an estimate of the expected performance $J(\boldsymbol{\omega}, \mathbf{c})$ in context $\mathbf{c}$.

Since $J(\boldsymbol{\omega}, \mathbf{c}) = \mathbb{E}_{p_{0,\mathbf{c}}(\mathbf{s}_0)}[V_{\boldsymbol{\omega}}(\mathbf{s}_0, \mathbf{c})]$, we can coarsely approximate the expectation w.r.t. $p_{0,\mathbf{c}}(\mathbf{s}_0)$ for a given context $\mathbf{c}_i$ with the initial state $\mathbf{s}_{i,0}$ contained in the set of trajectories $\mathcal{D}_k$. This yields an approximate form of (12) given by

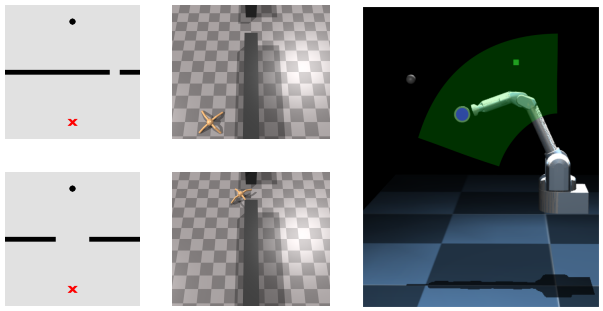
$$\max_{\boldsymbol{\nu}_{k+1}} \frac{1}{M} \sum_{i=1}^{M} \frac{p\left(\mathbf{c}_i|\boldsymbol{\nu}_{k+1}\right)}{p\left(\mathbf{c}_i|\boldsymbol{\nu}_k\right)} \tilde{V}_{\boldsymbol{\omega}}\left(\boldsymbol{s}_{i,0}, \mathbf{c}_i\right) - \alpha_k D_{\mathrm{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}_{k+1}) \parallel \mu(\mathbf{c})\right)$$
$$\text{s.t. } D_{\mathrm{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}_{k+1}) \parallel p(\mathbf{c}|\boldsymbol{\nu}_k)\right) \leq \epsilon. \tag{19}$$

The first term in objective (19) is an approximation to $\mathbb{E}_{p(\mathbf{c}|\boldsymbol{\nu}_{k+1})}[J(\boldsymbol{\omega}, \mathbf{c})]$ via importance-weights. The above objective can be solved using any constrained optimization algorithm. In our implementation, we use the trust-region algorithm implemented in the SciPy library (Virtanen et al., 2020). The two KL divergences in (19) can be computed in closed form since $\mu(\mathbf{c})$ and $p(\mathbf{c}|\boldsymbol{\nu})$ are Gaussians in our implementations. However, for more complicated distributions, the divergences can also be computed using samples from the respective distributions and the corresponding (unnormalized) log-likelihoods. The resulting approach (SPDL) is summarized in Algorithm 2.

7.2 Experiments

We evaluate SPDL in three different environments (Figure 7) with different deep RL (DRL) algorithms: TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017) and SAC (Haarnoja et al., 2018). For all DRL algorithms, we use the implementations from the `Stable Baselines` library (Hill et al., 2018). [2]

The first environment for testing SPDL is again a point-mass environment but with an additional parameter to the context space, as we will detail in the corresponding section. The second environment extends the point-mass experiment by replacing the point-mass with a torque-controlled quadruped 'ant', thus increasing the complexity of the underlying control problem and requiring the capacity of deep neural network function approximators used in DRL algorithms. Both of the mentioned environments will focus on learning a specific hard target task. 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 tar-



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

Figure 7: 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).

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

get 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 (Portelas et al., 2019), which is based on the concept of Intrinsic Motivation, and GoalGAN (Florensa et al., 2018), which relies on the notion of a success indicator to define a curriculum. Further, we also 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 the appendix.

7.2.1 Point-Mass Environment

As previously mentioned, we again focus on a point-mass environment, where now the control policy is a neural network. Furthermore, the contextual variable $\mathbf{c} \in \mathbb{R}^3$ now 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 $\mu(\mathbf{c})$ is a narrow Gaussian with a negligible variance that encodes a small gate at a specific position and a dynamic friction coefficient of 0. Figure 7 shows two different instances of the environment, one of them being the target task.
Figure 8 shows the results of two different experiments in this environment, one where the curriculum is generated over the full three-dimensional context space and one in which the friction parameter is fixed to its target value of 0 so that the curriculum is generated

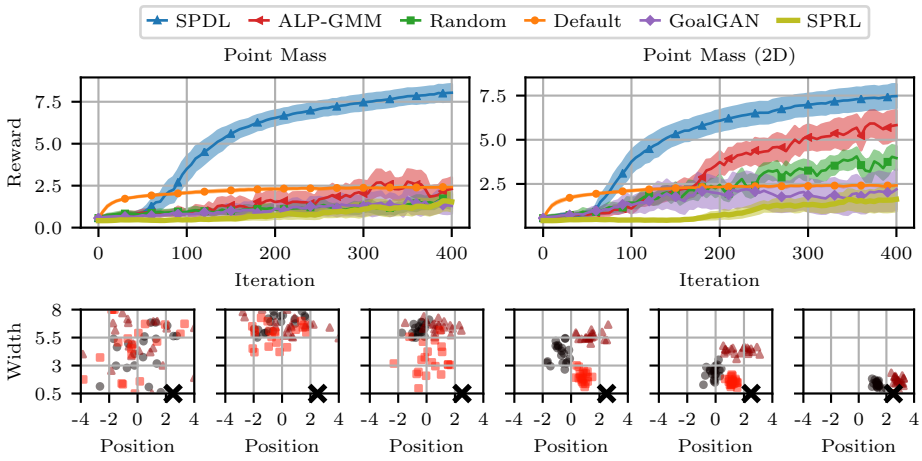


Figure 8: Reward of different curricula in the point-mass (2D and 3D) environment for TRPO. Mean (thick line) and two times standard error (shaded area) is computed from 20 algorithm runs. The lower plots show samples from the context distributions $p(\mathbf{c}|\boldsymbol{\nu})$ in the point-mass 2D environment at iterations 0, 20, 30, 50, 65 and 120 (from left to right). Different colors and shapes of samples indicate different algorithm runs. The black cross marks the mean of the target distribution $\mu(\mathbf{c})$.

only in a two-dimensional subspace. As Figure 8 and Table 1 indicate, SPDL significantly increases the asymptotic reward on the target task compared to other methods. Increasing the dimension of the context space harms the performance of the other CRL algorithms. For SPDL, there is no statistically significant difference in performance across the two settings. This observation is in line with the hypothesis posed in Section 6.2.1, that SPDL leverages the notion of $\mu(\mathbf{c})$ compared to other CRL algorithms that are not aware of it. As the context dimension increases, the volume of those parts in context space that carry non-negligible probability density according to $\mu(\mathbf{c})$ become smaller and smaller compared to the volume of the whole context space. Hence curricula that always target the whole context space tend to spend less time training on tasks that are relevant under $\mu(\mathbf{c})$. By having a notion of a target distribution, SPDL ultimately samples contexts that are likely according to $\mu(\mathbf{c})$, regardless of the dimension. The context distributions $p(\mathbf{c}|\boldsymbol{\nu})$ visualized in Figure 8 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 paces and in different ways, sometimes preferring to first shrink the width of the gate before moving its position while sometimes doing both simultaneously. More interestingly, the behavior of the curriculum is consistent with the one observed in Section 6.2.1. We further see that the episodic version (SPRL), which we applied by defining the episodic RL policy $p(\boldsymbol{\theta}|\mathbf{c}, \boldsymbol{\omega})$ to choose the weights $\boldsymbol{\theta}$ of a policy network for a given context $\mathbf{c}$, learns much slower compared to its step-based counterpart, requiring up to 800 iterations to reach an average reward of 5 (only the first 400 are shown in Figure 8). To keep the dimension of the context space moderate, the policy network for SPRL consisted of one layer of 21 tanh-activated hidden units, leading to 168 and 189 parameter dimensions in the two 2D and 3D context space instances. To make sure that the performance difference is not caused by different policy architectures, we also evaluated SPDL with this policy architecture, still significantly outperforming SPRL with an average reward of around 8 after 800 iterations.

7.2.2 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 (Brockman et al., 2016). [3] The goal is to reach the other side of a wall by passing through a gate, whose width and position are determined by the contextual variable $\mathbf{c} \in \mathbb{R}^2$ (see Figure 7). In this environment, we were only able to evaluate the CRL 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 the appendix).

Looking at Figure 9, 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 the appendix, both ALP-GMM and a random curriculum improve the qualitative performance, as they sometimes allow the ant to move through the gate. Nonetheless, this behavior is less efficient than the one learned by GoalGAN and SPDL, causing the action

3. We use the Nvidia Isaac Gym simulator (Nvidia, 2019) for this experiment.

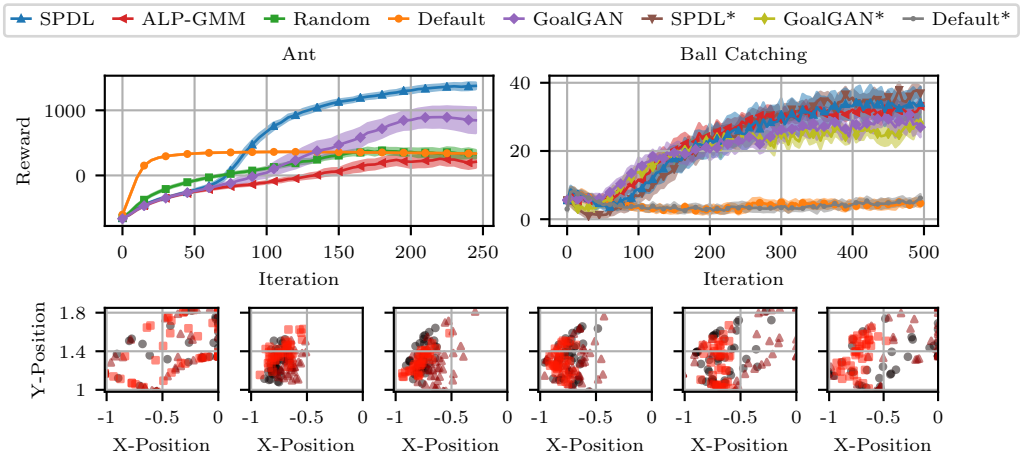


Figure 9: Mean (thick line) and two times standard error (shaded area) of the reward achieved with different curricula in the ant environment for PPO and in the ball-catching environment for SAC (upper plots). The statistics are computed from 20 seeds. For ball-catching, runs of SPDL/GoalGAN with an 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 $p(\mathbf{c}|\boldsymbol{\nu})$ 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 $p(\mathbf{c}|\boldsymbol{\nu})$ is initialized with $\mu(\mathbf{c})$, the samples in iteration 0 visualize the target distribution.

penalties in combination with the discount factor to prevent this better behavior from being reflected in the reward.

7.2.3 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 excessive movements. In the simulated environment, the ball is considered caught if it is in contact with the end effector. The context $\mathbf{c} \in \mathbb{R}^3$ 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 7 shows the robot as well as the target distribution over the ball positions in the aforementioned 'catching' plane. The context $\mathbf{c}$ 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 CRL algorithms.

	PPO (P3D)	SAC (P3D)	PPO (P2D)	SAC (P2D)	TRPO (BC)	PPO (BC)
ALP-GMM	2.43 ± 0.3	4.68 ± 0.8	5.23 ± 0.4	5.11 ± 0.7	39.8 ± 1.1	46.5 ± 0.7
GoalGAN	0.66 ± 0.1	2.14 ± 0.6	1.63 ± 0.5	1.34 ± 0.4	42.5 ± 1.6	42.6 ± 2.7
GoalGAN*	-	-	-	-	45.8 ± 1.0	45.9 ± 1.0
SPDL	$\mathbf{8.45 \pm 0.4}$	6.85 ± 0.8	$\mathbf{8.94 \pm 0.1}$	5.67 ± 0.8	47.0 ± 2.0	$\mathbf{53.9 \pm 0.4}$
SPDL*	-	-	-	-	43.3 ± 2.0	49.3 ± 1.4
Random	0.67 ± 0.1	2.70 ± 0.7	2.49 ± 0.3	4.99 ± 0.8	-	-
Default	2.40 ± 0.0	2.47 ± 0.0	2.37 ± 0.0	2.40 ± 0.0	21.0 ± 0.3	22.1 ± 0.3
Default*	-	-	-	-	21.2 ± 0.3	23.0 ± 0.7

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 Welch's 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.

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 9 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 slightly improves performance for GoalGAN while slightly reducing performance for SPDL. The context distributions $p(\mathbf{c}|\boldsymbol{\nu})$ visualized in Figure 9 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 progress takes place at a differing pace, as can be seen in the visualizations of $p(\mathbf{c}|\boldsymbol{\nu})$ in Figure 9 for iteration 200: Two of the three distributions fully match the target distribution while the third only covers half of it.

The similar performance across curriculum learning methods is indeed interesting. Clearly, the wide target context distribution $\mu(\mathbf{c})$ better matches the implicit assumptions made by both ALP-GMM and GoalGAN that learning should aim to accomplish tasks in the whole context space. However, both ALP-GMM and GoalGAN are built around the idea to sample tasks that promise a maximum amount of learning progress. This typically leads to a sampling scheme that avoids re-sampling tasks that the agent can already solve. However, SPDL achieves the same performance by simply growing the sampling distribution over time, not at all avoiding to sample tasks that the agent has mastered long ago. Hence, a

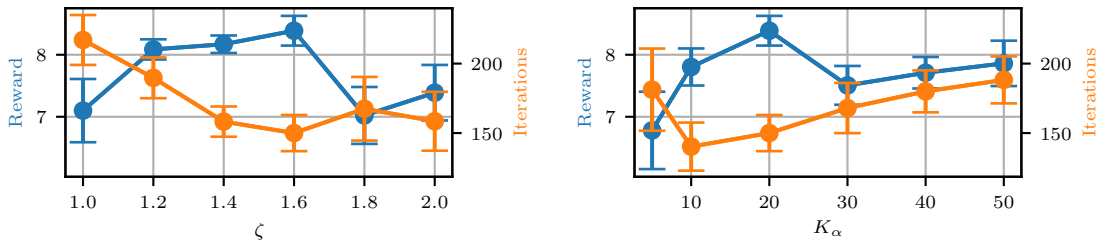


Figure 10: Final performance (blue) of SPDL on the point mass (3D) environment for different values of K_α and ζ as well as the average number of iterations required to reach a reward larger or equal to 5 on tasks sampled from $\mu(\mathbf{c})$ (orange). The results are computed from 20 environment runs. The error bars indicate the standard error. When varying ζ, K_α was fixed to a value of 20. When varying K_α, ζ was fixed to a value of 1.6.

promising direction for further improving the performance of CRL methods is to combine ideas of SPDL and methods such as ALP-GMM and GoalGAN.

8. Improved α-Schedule

The evaluation in the previous settings showed that choosing the trade-off parameter α_k in each iteration according to (13) was sufficient to improve the performance of the learner in the investigated experiments. However, the introduced schedule requires an offset parameter K_α as well as a penalty proportion ζ to be specified. Both these parameters need to be chosen adequately. If K_α is chosen too small, the agent may not have enough time to find a subspace of the context space containing tasks of adequate difficulty. If K_α is chosen too large, the learner wastes iterations focusing on tasks of small difficulty, not making progress towards the tasks likely under $\mu(\mathbf{c})$. The parameter ζ exhibits a similar trade-off behavior. Too small values will lead to an unnecessarily slow progression towards $\mu(\mathbf{c})$ while too large values lead to ignoring the competence of the learner on the tasks under $p(\mathbf{c}|\boldsymbol{\nu})$, resulting in a poor final agent behavior because of a too speedy progression towards $\mu(\mathbf{c})$. This trade-off is visualized in Figure 10 for the point mass environment. Despite the clear interpretation of the two parameters K_α and ζ, which allows for a fairly straightforward tuning, we additionally explore another approach of choosing α_k in this section. This approach only requires to specify an expected level of performance V_{LB} that the agent should maintain under the chosen context distribution $p(\mathbf{c}|\boldsymbol{\nu})$. Assuming the decoupled optimization of the policy π and the context distribution $p(\mathbf{c}|\boldsymbol{\nu})$ investigated in the last section, this can be easily realized by rewriting (12) as

$$\begin{aligned} &\min_{\boldsymbol{\nu}} \; D_{\text{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}) \parallel \mu(\mathbf{c})\right) \\ &\text{s.t. } \mathbb{E}_{p(\mathbf{c}|\boldsymbol{\nu})}\left[J(\boldsymbol{\omega}, \mathbf{c})\right] \geq V_{\text{LB}} \\ &\quad D_{\text{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}) \parallel p(\mathbf{c}|\boldsymbol{\nu}')\right) \leq \epsilon. \end{aligned} \tag{20}$$

The main modification is to avoid the explicit trade-off between expected agent performance and KL divergence by minimizing the KL divergence w.r.t. $\mu(\mathbf{c})$ subject to a constraint on the expected agent performance. Investigating the Lagrangian of (20)

$$\begin{aligned} L(\boldsymbol{\nu}, \alpha, \eta) = D_{\mathrm{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}) \parallel \mu(\mathbf{c})\right) + \alpha(V_{\mathrm{LB}} - \mathbb{E}_{p(\mathbf{c}|\boldsymbol{\nu})}\left[J(\boldsymbol{\omega}, \mathbf{c})\right]) \\ + \eta(D_{\mathrm{KL}}\left(p(\mathbf{c}|\boldsymbol{\nu}) \parallel p(\mathbf{c}|\boldsymbol{\nu}')\right) - \epsilon), \qquad \alpha, \eta \geq 0 \end{aligned}$$

we see that the constraint reintroduces a scalar α that trades-off the expected agent performance and KL divergence to $\mu(\mathbf{c})$. The value of this scalar is, however, now automatically chosen to fulfill the imposed constraint on the expected agent performance. For an implementation of (20), we again replace $J(\boldsymbol{\omega}, \mathbf{c})$ by an importance-sampled Monte Carlo estimate as done in (19). At this point, we have replaced the parameter ζ with V_{LB}. The benefit is a more intuitive choice of this hyperparameter since it is directly related to the expected performance, i.e. the quantity being optimized. Furthermore, we can easily remove the need for the offset parameter K_α by setting α_k=0 in (19) until we first reach V_{LB}. Consequently, this new schedule only requires one hyperparameter V_{LB} to be specified. From then on, we compute the new context distribution by optimizing (20). If during learning, the performance of the agent falls below V_{LB} again, we simply do not change the context distribution until the performance exceeds V_{LB} again. We now compare this new schedule with the schedule for α that is based on K_α and ζ. The corresponding experiment data is shown in Table 2. We can see that the final rewards achieved with the two heuristics

	SPDL(K_α, ζ)		SPDL(V_{LB})	
	Performance	Iterations to Threshold	Performance	Iterations to Threshold
TRPO (P3D)	8.04 ± 0.25	198 ± 18	7.79 ± 0.28	220 ± 19
PPO (P3D)	8.45 ± 0.42	165 ± 14	8.66 ± 0.07	$\mathbf{120 \pm 10}$
SAC (P3D)	6.85 ± 0.77	94 ± 8	7.13 ± 0.71	$\mathbf{67 \pm 4}$
TRPO (P2D)	7.47 ± 0.33	201 ± 20	7.67 ± 0.2	198 ± 19
PPO (P2D)	8.94 ± 0.10	132 ± 5	9.01 ± 0.07	$\mathbf{119 \pm 3}$
SAC (P2D)	5.67 ± 0.77	134 ± 30	6.56 ± 0.82	$\mathbf{59 \pm 3}$
PPO (ANT)	1371 ± 23	131 ± 3	1305 ± 38	131 ± 2
TRPO (BC)	47.0 ± 2.0	379 ± 21	50.0 ± 1.5	320 ± 20
PPO (BC)	53.9 ± 0.4	285 ± 19	51.6 ± 1.7	$\mathbf{234 \pm 12}$
SAC (BC)	34.1 ± 2.3	205 ± 12	34.1 ± 1.3	$\mathbf{139 \pm 8}$
TRPO (BC*)	43.3 ± 2.0	354 ± 18	46.0 ± 1.5	$\mathbf{285 \pm 20}$
PPO (BC*)	49.3 ± 1.4	224 ± 7	51.8 ± 0.5	212 ± 17
SAC (BC*)	36.9 ± 1.0	235 ± 23	37.1 ± 1.2	$\mathbf{173 \pm 20}$

Table 2: Comparison between the two SPDL heuristics on the point-mass (P3D and P2D), ant and ball-catching (BC) environments computed using 20 runs. The asterisks mark runs of SPDL with an initialized context distribution. We compare both the final average reward $\pm$ standard error (Performance) as well as the average number of iterations required to reach 80% of the lower of the two rewards (Iterations to Threshold). Statistically significant differences according to Welch's t-test are highlighted in **bold** for p<1% and **brown** for p<5%.

are not significantly different according to Welch's t-test. This indicates that the simpler heuristic performs just as well in the investigated environments in terms of final reward. However, the often significantly smaller average number of iterations required to reach a certain average performance on $\mu(\mathbf{c})$ shows that the schedule based on V_{LB} tends to lead to a faster progression towards $\mu(\mathbf{c})$. In a sense, this is not surprising, given that the explicit minimization of the KL divergence w.r.t. $\mu(\mathbf{c})$ under a constraint on the expected performance optimizes the trade-off between agent performance and KL divergence to $\mu(\mathbf{c})$ in each iteration. With the schedule based on K_α and ζ, this trade-off was compressed into the parameter ζ that stayed constant for all iterations. Furthermore, the lower bound V_{LB} on the achieved average reward under $p(\mathbf{c}|\boldsymbol{\nu})$ exhibits certain similarities to the GoalGAN algorithm in which the context distribution resulted from a constraint of encoding only tasks of intermediate difficulty. In a sense, GoalGAN uses an upper and lower bound on the task difficulty. However, it enforces this constraint per context while our formulation enforces the lower bound in expectation.

9. An Inference Perspective on Self-Paced Reinforcement Learning

As noted in Section 5, the KL divergence regularization w.r.t. $\boldsymbol{\nu}$ in (12) was done to stabilize the overall learning procedure. In this final section, we show that the resulting learning scheme can be connected to a modified version of the expectation-maximization algorithm, a well-known majorize-minimize algorithm for inference problems. Before we conclude this paper, we want to briefly point out this connection, especially highlighting a connection between self-paced learning and the concept of tempering (Kirkpatrick et al., 1983; Van Laarhoven and Aarts, 1987).

9.1 RL as Inference

To establish the aforementioned connection, we need to introduce a probabilistic interpretation of the contextual RL problem as being an inference task (Dayan and Hinton, 1997; Toussaint and Storkey, 2006; Levine, 2018). In this formulation, the goal is to maximize the probability of an optimality event $\mathcal{O} \in \{0, 1\}$ that depends on the sum of rewards along a trajectory of states and actions $\boldsymbol{\tau} = \{(\mathbf{s}_t, \mathbf{a}_t)|t = 0, 1, \ldots\}$

$$p(\mathcal{O}|\boldsymbol{\tau}, \mathbf{c}) \propto \exp\left(r(\boldsymbol{\tau}, \mathbf{c})\right) = \exp\left(\sum_{t=0}^{\infty} r_{\mathbf{c}}(\mathbf{s}_t, \mathbf{a}_t)\right). \tag{21}$$

Together with the probability for a trajectory $\boldsymbol{\tau}$ given the policy parameters $\boldsymbol{\omega}$

$$p(\boldsymbol{\tau}|\mathbf{c}, \boldsymbol{\omega}) = p_{0,\mathbf{c}}(\boldsymbol{s}_0) \prod_{t \geq 0} \bar{p}_{\mathbf{c}}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{s}_t|\mathbf{a}_t, \mathbf{c}, \boldsymbol{\omega}) \tag{22}$$

we can marginalize over the trajectories $\boldsymbol{\tau}$ generated by $p(\boldsymbol{\tau}|\mathbf{c}, \boldsymbol{\omega})$, i.e. generated by the agent. This results in a probabilistic equivalent of the expected performance $J(\boldsymbol{\omega}, \mathbf{c})$ in context $\mathbf{c}$ under policy parameters $\boldsymbol{\omega}$. This probabilistic equivalent is given by the marginal likelihood of event $\mathcal{O}$ in an MDP $\mathcal{M}(\mathbf{c})$

$$J(\boldsymbol{\omega}, \mathbf{c}) = p(\mathcal{O}|\mathbf{c}, \boldsymbol{\omega}) = \int p(\mathcal{O}|\boldsymbol{\tau}, \mathbf{c})p(\boldsymbol{\tau}|\mathbf{c}, \boldsymbol{\omega})\, \mathrm{d}\boldsymbol{\tau}. \tag{23}$$

The transition probabilities $\bar{p}_{\mathbf{c}}$ in (22) are a modified version of the original transition probabilities $p_{\mathbf{c}}$ that introduce a "termination" probability in each step that can occur with a probability of $1-\gamma$ (see Levine, 2018). This introduces the concept of a discounting factor γ into the probabilistic model. Introducing a context distribution $p(\mathbf{c}|\boldsymbol{\nu})$ then yields a probabilistic interpretation of the contextual RL objective

$$J(\boldsymbol{\omega}, p(\mathbf{c}|\boldsymbol{\nu})) = p(\mathcal{O}|\boldsymbol{\omega}, \boldsymbol{\nu}) = \int p(\mathcal{O}|\boldsymbol{\tau}, \mathbf{c}) p(\boldsymbol{\tau}|\mathbf{c}, \boldsymbol{\omega}) p(\mathbf{c}|\boldsymbol{\nu}) \, \mathrm{d}\mathbf{c} \, \mathrm{d}\boldsymbol{\tau}. \tag{24}$$

When not making use of a curriculum, we would simply set $p(\mathbf{c}|\boldsymbol{\nu}) = \mu(\mathbf{c})$. The above model is called a latent variable model (LVM), as the trajectories $\boldsymbol{\tau}$, as well as the contexts $\mathbf{c}$ are marginalized out to form the likelihood of the event $\mathcal{O}$. These marginalizations make the direct optimization w.r.t. $\boldsymbol{\omega}$ and $\boldsymbol{\nu}$ challenging. The so-called expectation-maximization algorithm is commonly applied to split this complicated optimization into two simpler steps: The E- and M-Step

$$\text{E-Step}: q^k(\boldsymbol{\tau}, \mathbf{c}) = \arg\min_{q(\boldsymbol{\tau}, \mathbf{c})} \; D_{\mathrm{KL}} \left(q(\boldsymbol{\tau}, \mathbf{c}) \;\middle\|\; p(\boldsymbol{\tau}, \mathbf{c}|\mathcal{O}, \boldsymbol{\omega}^k, \boldsymbol{\nu}^k) \right) \tag{25}$$

$$\text{M-Step}: \boldsymbol{\omega}^{k+1}, \boldsymbol{\nu}^{k+1} = \arg\max_{\boldsymbol{\omega}, \boldsymbol{\nu}} \; \mathbb{E}_{q^k(\boldsymbol{\tau}, \mathbf{c})} \left[\log(p(\mathcal{O}, \boldsymbol{\tau}, \mathbf{c}|\boldsymbol{\omega}, \boldsymbol{\nu})) \right]. \tag{26}$$

Iterating between these two steps is guaranteed to find a local optimum of the marginal likelihood $p(\mathcal{O}|\boldsymbol{\omega}, \boldsymbol{\nu})$.

9.2 Connection to Self-Paced Reinforcement Learning

At this point, two simple reformulations are required to establish the connection between the KL-regularized objective (12) and the expectation-maximization algorithm on LVM (24). First, we can reformulate the M-Step as an M-projection (i.e. a maximum-likelihood fit of the parametric model $q(\boldsymbol{\tau}, \mathbf{c}|\boldsymbol{\omega}, \boldsymbol{\nu})$ to $q^k(\boldsymbol{\tau}, \mathbf{c})$)

$$\arg\max_{\boldsymbol{\omega}, \boldsymbol{\nu}} \; \mathbb{E}_{q^k(\boldsymbol{\tau}, \mathbf{c})} \left[\log(p(\mathcal{O}, \boldsymbol{\tau}, \mathbf{c}|\boldsymbol{\omega}, \boldsymbol{\nu})) \right] = \arg\min_{\boldsymbol{\omega}, \boldsymbol{\nu}} D_{\mathrm{KL}} \left(q^k(\boldsymbol{\tau}, \mathbf{c}) \;\middle\|\; q(\boldsymbol{\tau}, \mathbf{c}|\boldsymbol{\omega}, \boldsymbol{\nu}) \right).$$

Second, the E-Step can, for this particular model, be shown to be equivalent to a KL-regularized RL objective

$$\begin{aligned} & \arg\min_{q(\boldsymbol{\tau}, \mathbf{c})} \; D_{\mathrm{KL}} \left(q(\boldsymbol{\tau}, \mathbf{c}) \;\middle\|\; p(\boldsymbol{\tau}, \mathbf{c}|\mathcal{O}, \boldsymbol{\omega}^k, \boldsymbol{\nu}^k) \right) \\ = \; & \arg\max_{q(\boldsymbol{\tau}, \mathbf{c})} \mathbb{E}_{q(\boldsymbol{\tau}, \mathbf{c})} \left[R(\boldsymbol{\tau}, \mathbf{c}) \right] - D_{\mathrm{KL}} \left(q(\boldsymbol{\tau}, \mathbf{c}) \parallel p(\boldsymbol{\tau}, \mathbf{c}|\boldsymbol{\omega}, \boldsymbol{\nu}) \right), \end{aligned}$$

in which we penalize a deviation of the policy and context distribution from the current parametric distribution $p(\boldsymbol{\tau}, \mathbf{c}|\boldsymbol{\omega}, \boldsymbol{\nu})$. Adding a term $-\alpha D_{\mathrm{KL}} \left(q(\mathbf{c}) \parallel \mu(\mathbf{c}) \right)$ and optimizing this modified E-Step only w.r.t. the context distribution $q(\mathbf{c})$ while keeping $q(\boldsymbol{\tau}|\mathbf{c})$ fixed at $p(\boldsymbol{\tau}|\mathbf{c}, \boldsymbol{\omega}^k)$, we obtain

$$\arg\max_{q(\mathbf{c})} \mathbb{E}_{q(\mathbf{c})} \left[\mathbb{E}_{p(\boldsymbol{\tau}|\mathbf{c}, \boldsymbol{\omega}^k)} \left[R(\boldsymbol{\tau}, \mathbf{c}) \right] \right] - \alpha D_{\mathrm{KL}} \left(q(\mathbf{c}) \parallel \mu(\mathbf{c}) \right) - D_{\mathrm{KL}} \left(q(\mathbf{c}) \;\middle\|\; p(\mathbf{c}|\boldsymbol{\nu}^k) \right). \tag{27}$$

This result resembles (12), where however the optimization is carried out w.r.t. $q(\mathbf{c})$ instead of $\boldsymbol{\nu}$ and the KL divergence w.r.t. $p(\mathbf{c}|\boldsymbol{\nu}^k)$ is treated as a penalty term instead of a constraint. Not fitting the parameters $\boldsymbol{\nu}^{k+1}$ directly but in a separate (M-)step is also done by C-REPS and our episodic RL implementation of SPL. Hence, in the light of these results, the step-based implementation can be interpreted as skipping an explicit M-Step and directly optimizing the E-Step w.r.t. to the parametric policy. Such a procedure can be found in popular RL algorithms, as detailed by Abdolmaleki et al. (2018).

9.3 Self-Paced Learning as Tempering

The previously derived E-Step has a highly interesting connection to a concept in the inference literature called *tempering* (Kirkpatrick et al., 1983; Van Laarhoven and Aarts, 1987). This connection is revealed by showing that the penalty term $\alpha D_{\mathrm{KL}}\left(q(\mathbf{c}) \parallel \mu(\mathbf{c})\right)$ in the modified E-Step (27) results in an E-Step to a modified target distribution. That is

$$\begin{aligned} &\arg\min_{q(\mathbf{c})} \; D_{\mathrm{KL}}\left(q(\mathbf{c}) \;\middle\|\; p(\mathbf{c}|\mathcal{O}, \boldsymbol{\omega}^k, \boldsymbol{\nu}^k)\right) + \alpha D_{\mathrm{KL}}\left(q(\mathbf{c}) \parallel \mu(\mathbf{c})\right) \\ = \; &\arg\min_{q(\mathbf{c})} \; D_{\mathrm{KL}}\left(q(\mathbf{c}) \;\middle\|\; \frac{1}{Z} p(\mathbf{c}|\mathcal{O}, \boldsymbol{\omega}^k, \boldsymbol{\nu}^k)^{\frac{1}{1+\alpha}} \mu(\mathbf{c})^{\frac{\alpha}{1+\alpha}}\right). \end{aligned} \tag{28}$$

The modified target distribution in (28) is performing an interpolation between $\mu(\mathbf{c})$ and $p(\mathbf{c}|\mathcal{O}, \boldsymbol{\omega}^k, \boldsymbol{\nu}^k)$ based on the parameter α. Looking back at the sampling distribution induced by the probabilistic SPL objective (9) for the regularizer $f_{\mathrm{KL},i}$ (see Equation 30 in the appendix)

$$p(\mathbf{c}|\alpha, \boldsymbol{\omega}) \propto \boldsymbol{\nu}^*_{\mathrm{KL},\mathbf{c}}(\alpha, \boldsymbol{\omega}) = \mu(\mathbf{c}) \exp\left(-J(\boldsymbol{\omega}, \mathbf{c})\right)^{\frac{1}{\alpha}}, \tag{29}$$

we can see that, similarly to the modified E-Step, the distribution encoded by $\boldsymbol{\nu}^*_{\mathrm{KL},\mathbf{c}}$, i.e. the optimizers of (9), interpolates between $\mu(\mathbf{c})$ and the distribution $p(\mathbf{c}|\boldsymbol{\omega}) \propto \exp\left(-J(\boldsymbol{\omega}, \mathbf{c})\right)$. Both of these distributions would be referred to as tempered distributions in the inference literature.

The concept of tempering has been explored in the inference literature as a tool to improve inference methods when sampling from or finding modes of a distribution $\mu(\mathbf{c})$ with many isolated modes of density (Kirkpatrick et al., 1983; Marinari and Parisi, 1992; Ueda and Nakano, 1995). The main idea is to not directly apply inference methods to $\mu(\mathbf{c})$ but to make use of a tempered distribution $p_\alpha(\mathbf{c})$ which interpolates between $\mu(\mathbf{c})$ and a user-chosen reference distribution $\rho(\mathbf{c})$ from which samples can be easily drawn by the employed inference method (e.g. a Gaussian distribution). Doing repeated inference for varying values of α allows to explore the isolated modes more efficiently and with that yielding more accurate samples from $\mu(\mathbf{c})$. Intuitively, initially sampling from $\rho(\mathbf{c})$, chosen to be free from isolated modes, and gradually progressing towards $\mu(\mathbf{c})$ while using the previous inferences as initializations avoids getting stuck in isolated modes of $\mu(\mathbf{c})$ that encode comparatively low density. This technique makes the inference algorithm less dependent on a good initialization.

We can easily identify both (28) and (29) to be particular tempered distributions $p_\alpha(\mathbf{c})$. There, however, seems to be a striking difference to the aforementioned tempering scheme:

The target density $\mu(\mathbf{c})$ is typically trivial, not requiring any advanced inference machinery. However, although $\mu(\mathbf{c})$ may be trivial from an inference perspective, the density $p(\boldsymbol{\omega}|\mathcal{O}) \propto p(\boldsymbol{\omega}) \int p(\mathcal{O}|\mathbf{c}, \boldsymbol{\omega})\mu(\mathbf{c})\, \mathrm{d}\mathbf{c}$, i.e. the posterior over policy parameters, is highly challenging for contexts $\mathbf{c}$ distributed according to $\mu(\mathbf{c})$. This is because it may contain many, highly isolated modes, many of which only encode suboptimal behavior. In these cases, tempering helps to achieve better performance when employed in combination with RL. For low values of α, it is easier to find high-density modes of $p_\alpha(\boldsymbol{\omega}|\mathcal{O}) \propto p(\boldsymbol{\omega}) \int p(\mathcal{O}|\mathbf{c}, \boldsymbol{\omega}) p_\alpha(\mathbf{c}) d\mathbf{c}$. These modes can then be "tracked" by the RL algorithm while increasing the value of α.
The connection between SPL and the concept of tempering yields interesting insights into the problem of choosing both a good schedule for α and also the general design of $p_\alpha(\mathbf{c})$. As introduced in Section 3.1, the particular choice of the self-paced regularizer $f(\alpha, \boldsymbol{\nu})$, and hence the regularizer $F_\alpha(l)$, is closely related to the particular form of $p_\alpha(\mathbf{c})$. A ubiquitous decision is the choice of the particular regularizer or tempered distribution for a given problem. Gelman and Meng (1998) show that the particular choice of p_α has a tremendous effect on the error of Monte Carlo estimates of ratios between normalization constants. Furthermore, they compute the optimal form of p_α for a Gaussian special case that reduces the variance of the Monte Carlo estimator. It may be possible to draw inspiration from their techniques to design regularizers specialized for problems that fulfill particular properties.
For the application of SPL to RL, another important design decision is the schedule of α. The value of α should be increased as fast as possible while ensuring the stability of the RL agent. We proposed two schedules that accomplished this task sufficiently well. However, there may be a tremendous margin for improvement. In the inference literature, people readily investigated the problem of choosing α, as they face a similar trade-off problem between required computation time of inference methods and the usefulness of their results (Mandt et al., 2016; Graham and Storkey, 2017; Luo et al., 2018). Again, it may be possible to draw inspiration from these works to design better schedules for α in RL problems.

10. Conclusion and Discussion

We have presented an interpretation of self-paced learning as inducing a sampling distribution over tasks in a reinforcement learning setting when using the KL divergence w.r.t. a target distribution $\mu(\mathbf{c})$ as a self-paced regularizer. This view renders the induced curriculum as an approximate implementation of a regularized contextual RL objective that samples training tasks based on their contribution to the overall gradient of the objective. Furthermore, we identified our approximate implementations to be a modified version of the expectation-maximization algorithm applied to the common latent variable model for RL. These, in turn, revealed connections to the concept of tempering in the inference literature. These observations motivate further theoretical investigations, such as identifying the particular regularized objective that is related to our approximate implementation (6). Furthermore, we only explored the KL divergence as a self-paced regularizer. Although we showed that the probabilistic interpretation of SPL does not hold for arbitrary regularizers, it may be possible to derive the presented results for a wider class of regularizers, such as f-divergences.
From an experimental point of view, we focused particularly on RL tasks with a continuous context space in this work. In the future, we want to conduct experiments in discrete

context spaces, where we do not need to restrict the distribution to some tractable analytic form since we can exactly represent discrete probability distributions.
Our implementations of the SPL scheme for RL demonstrated remarkable performance across RL algorithms and tasks. The presented algorithms are, however, by far no perfect realizations of the theoretical concept. The proposed ways of choosing α in each iteration are just ad-hoc choices. At this point, insights gained through the inference perspective into our curriculum generation scheme presented in Section 9 may be particularly useful. Furthermore, the use of Gaussian context distributions is a major limitation that restricts the flexibility of the context distribution. Specifically in higher-dimensional context spaces, such a restriction could lead to poor performance. Here, it may be possible to use advanced inference methods (Liu et al., 2019; Wibisono, 2018) to sample from the distribution (28) without approximations even in continuous spaces.

Acknowledgments

This project has received funding from the DFG project PA3179/1-1 (ROBOLEAP) and from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 640554 (SKILLS4ROBOTS). Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt.

Appendix A. Proof of Theorem 1

We begin by restating the theorem from the main text

Theorem 1 *Alternatingly solving*

$$\min_{\boldsymbol{\omega},\boldsymbol{\nu}} \mathbb{E}_{p(c|\boldsymbol{\nu})}\left[l(\mathbf{x}_c, y_c, \boldsymbol{\omega})\right] + \alpha D_{KL}\left(p(c|\boldsymbol{\nu}) \parallel \mu(c)\right)$$

w.r.t. $\boldsymbol{\omega}$ *and* $\boldsymbol{\nu}$ *is a majorize-minimize scheme applied to the regularized objective*

$$\min_{\boldsymbol{\omega}} \mathbb{E}_{\mu(c)}\left[\alpha\left(1 - \exp\left(-\frac{1}{\alpha} l(\mathbf{x}_c, y_c, \boldsymbol{\omega})\right)\right)\right].$$

Proof To prove the theorem, we make use of the result established by Meng et al. (2017) that optimizing the SPL objective alternatingly w.r.t. $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$

$$\boldsymbol{\nu}^*, \boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\nu},\boldsymbol{\omega}} \; r(\boldsymbol{\omega}) + \sum_{i=1}^{N} \left(\nu_i l(\mathbf{x}_i, y_i, \boldsymbol{\omega}) + f(\alpha, \nu_i)\right), \qquad \alpha > 0. \tag{1}$$

is a majorize-minimize scheme applied to the objective

$$\min_{\boldsymbol{\omega}} r(\boldsymbol{\omega}) + \sum_{i=1}^{N} F_\alpha(l(\mathbf{x}_i, y_i, \boldsymbol{\omega})), \qquad F_\alpha(l(\mathbf{x}_i, y_i, \boldsymbol{\omega})) = \int_0^{l(\mathbf{x}_i, y_i, \boldsymbol{\omega})} \nu^*(\alpha, \iota)\, \mathrm{d}\iota. \tag{3}$$

Based on this result, the proof of Theorem 1 requires three steps: **First**, we need to show that the function

$$f_{\mathrm{KL},i}(\alpha, \nu) = \alpha\nu\left(\log(\nu) - \log(\mu(c{=}i))\right) - \alpha\nu, \tag{10}$$

is a valid self-paced regularizer for objective (1) and that the corresponding objective (3) has the form of the second objective in Theorem 1. **Second**, we need to show the equivalence between the SPL objective (1) and the probabilistic objective (9) for the regularizer $f_{\mathrm{KL},i}$. **Finally**, we need to show that objective (9) corresponds to the first objective in Theorem 1 when using $f_{\mathrm{KL},i}$. We begin by restating the axioms of self-paced regularizers defined by Jiang et al. (2015) to prove the first of the three points. Again making use of the notation $\nu^*(\alpha, l) = \arg\min_\nu \nu l + f(\alpha, \nu)$, these axioms are

1. $f(\alpha, \nu)$ is convex w.r.t. ν
2. $\nu^*(\alpha, l)$ is monotonically decreasing w.r.t. l and it holds that $\lim_{l\to 0} \nu^*(\alpha, l) = 1$ as well as $\lim_{l\to\infty} \nu^*(\alpha, l) = 0$
3. $\nu^*(\alpha, l)$ is monotonically decreasing w.r.t. α and it holds that $\lim_{\alpha\to\infty} \nu^*(\alpha, l) \leq 1$ as well as $\lim_{\alpha\to 0} \nu^*(\alpha, l) = 0$.

It is important to note that, due to the term $\mu(c{=}i)$ in (10), there is now an individual regularizer $f_{\mathrm{KL},i}$ for each sample. This formulation is in line with the theory established by Meng et al. (2017) and simply corresponds to an individual regularizer $F_{\alpha,i}$ for each sample in (3). Inspecting the second derivative of $f_{\mathrm{KL},i}$ w.r.t. ν, we see that $f_{\mathrm{KL},i}(\alpha, \nu)$ is convex w.r.t. ν. Furthermore, the solution to the SPL objective (1)

$$\nu^*_{\mathrm{KL},i}(\alpha, l) = \mu(c{=}i) \exp\left(-\frac{1}{\alpha} l\right) \tag{30}$$

fulfills above axioms except for $\lim_{l\to 0} \nu^*_{\mathrm{KL},i}(\alpha, l) = 1$, since $\lim_{l\to 0} \nu^*_{\mathrm{KL},i}(\alpha, l) = \mu(c{=}i)$. However, we could simply remove the log-likelihood term $\log(\mu(c{=}i))$ from $f_{\mathrm{KL},i}(\alpha, \nu_i)$ and pre-weight each sample with $\mu(c{=}i)$, which would yield exactly the same curriculum while fulfilling all axioms. We stick to the introduced form, as it eases the connection of $f_{\mathrm{KL},i}$ to the KL divergence between $p(c|\boldsymbol{\nu})$ and $\mu(c)$. Given that we have ensured that $f_{\mathrm{KL},i}$ is a valid self-paced regularizer, we know that optimizing the SPL objective (1) under $f_{\mathrm{KL},i}$ corresponds to employing the non-convex regularizer

$$F_{\mathrm{KL},\alpha,i}(l(\mathbf{x}_i, y_i, \boldsymbol{\omega})) = \int_0^{l(\mathbf{x}_i, y_i, \boldsymbol{\omega})} \nu^*_{\mathrm{KL},i}(\alpha, \iota)\, \mathrm{d}\iota = \mu(c{=}i)\alpha \left(1 - \exp\left(-\frac{1}{a} l(\mathbf{x}_i, y_i, \boldsymbol{\omega})\right)\right). \tag{31}$$

Put differently, optimizing the SPL objective (1) with $r(\boldsymbol{\omega}) = 0$ under $f_{\mathrm{KL},i}$ corresponds to optimizing

$$\min_{\boldsymbol{\omega}} \sum_{i=1}^{N} F_{\mathrm{KL},\alpha,i}(l(\mathbf{x}_i, y_i, \boldsymbol{\omega})) = \min_{\boldsymbol{\omega}} \mathbb{E}_{\mu(c)} \left[\alpha \left(1 - \exp\left(-\frac{1}{a} l(\mathbf{x}_c, y_c, \boldsymbol{\omega})\right)\right)\right],$$

as stated in Theorem 1. As a next step, we notice that entries in the optimal $\boldsymbol{\nu}$ for a given $\boldsymbol{\omega}$ and α in the probabilistic SPL objective (9) are proportional to $\nu^*_{\mathrm{KL},i}(\alpha, l)$ in (30), where the factor of proportionality Z simply rescales the variables $\nu^*_{\mathrm{KL},i}$ so that they fulfill the normalization constraint of objective (9). Since

$$\mathbb{E}_{p(c|\boldsymbol{\nu})} \left[f(\mathbf{x}_c, y_c, \boldsymbol{\omega})\right] = \sum_{i=1}^{N} \nu_i f(\mathbf{x}_i, y_i, \boldsymbol{\omega})$$

by definition of $p(c|\boldsymbol{\nu})$ introduced in Section 4, we see that consequently the only difference between (1) and (9) for this particular regularizer is a different weighting of the regularization term $r(\boldsymbol{\omega})$ throughout the iterations of SPL. More precisely, $r(\boldsymbol{\omega})$ is weighted by the aforementioned factor of proportionality Z. Since $r(\boldsymbol{\omega}) = 0$ in Theorem 1, SPL (1) and the probabilistic interpretation introduced in this paper (9) are exactly equivalent, since a constant scaling does not change the location of the optima w.r.t $\boldsymbol{\omega}$ in both (1) and (9). Consequently, we are left with proving that the PSPL objective (9) under $f_{\mathrm{KL},i}$ and $r(\boldsymbol{\omega}) = 0$ is equal to the first objective in Theorem 1. This reduces to proving that $\sum_{i=1}^{N} f_{\mathrm{KL},i}(\alpha, \nu_i)$ is equal to the KL divergence between $p(c|\boldsymbol{\nu})$ and $\mu(c)$. Remembering $p(c{=}i|\boldsymbol{\nu}) = \nu_i$, it follows that

$$\begin{aligned}\sum_{i=1}^{N} f_{\mathrm{KL},i}(\alpha, \nu_i) &= \alpha \sum_{i=1}^{N} p(c{=}i|\boldsymbol{\nu}) \left(\log(p(c{=}i|\boldsymbol{\nu})) - \log(\mu(c{=}i))\right) - \alpha \sum_{i=1}^{N} p(c{=}i|\boldsymbol{\nu}) \\ &= \alpha D_{\mathrm{KL}}\left(p(c{=}i|\boldsymbol{\nu}) \parallel \mu(c{=}i)\right) - \alpha.\end{aligned}$$

The removal of the sum in the second term is possible because $\sum_{i=1}^{N} p(c{=}i|\boldsymbol{\nu}) = 1$ per definition of a probability distribution. Since the constant value α does not change the optimization w.r.t. $\boldsymbol{\nu}$, this proves the desired equivalence and with that Theorem 1. ■

Appendix B. Self-Paced Episodic Reinforcement Learning Derivations

This appendix serves to highlight some important details regarding the derivation of the weights (16) and (17) as well as the dual objective (18). The most notable detail is the introduction of an additional distribution $q(\mathbf{c})$ that takes the role of the marginal $\int q(\boldsymbol{\theta}, \mathbf{c})\, \mathrm{d}\boldsymbol{\theta}$ as well as the regularization of this additional distribution via a KL divergence constraint w.r.t. to the previous marginal $p(\mathbf{c}) = \int p(\boldsymbol{\theta}, \mathbf{c})\, \mathrm{d}\boldsymbol{\theta}$. This yields the following objective

$$\begin{aligned}\max_{q(\boldsymbol{\theta},\mathbf{c}),q(\mathbf{c})} \; & \mathbb{E}_{q(\boldsymbol{\theta},\mathbf{c})}\left[r\left(\boldsymbol{\theta}, \mathbf{c}\right)\right] - \alpha D_{\mathrm{KL}}\left(q(\mathbf{c}) \parallel \mu(\mathbf{c})\right) \\ \text{s.t. } & D_{\mathrm{KL}}\left(q(\boldsymbol{\theta}, \mathbf{c}) \parallel p(\boldsymbol{\theta}, \mathbf{c})\right) \leq \epsilon \qquad \int q(\boldsymbol{\theta}, \mathbf{c})\, \mathrm{d}\mathbf{c}\, \mathrm{d}\boldsymbol{\theta} = 1 \\ & D_{\mathrm{KL}}\left(q(\mathbf{c}) \parallel p(\mathbf{c})\right) \leq \epsilon \qquad \int q(\mathbf{c})\, \mathrm{d}\mathbf{c} = 1 \\ & \int q(\boldsymbol{\theta}, \mathbf{c})\, \mathrm{d}\boldsymbol{\theta} = q(\mathbf{c}) \quad \forall \mathbf{c} \in \mathcal{C}.\end{aligned}$$

However, these changes are purely of technical nature as they allow to derive numerically stable weights and duals. It is straightforward to verify that $D_{\mathrm{KL}}\left(q(\boldsymbol{\theta}, \mathbf{c}) \parallel p(\boldsymbol{\theta}, \mathbf{c})\right) \leq \epsilon$ implies $D_{\mathrm{KL}}\left(q(\mathbf{c}) \parallel p(\mathbf{c})\right) \leq \epsilon$. Hence, the constraint $\int q(\boldsymbol{\theta}, \mathbf{c})\, \mathrm{d}\boldsymbol{\theta} = q(\mathbf{c})$ guarantees that a solution $q(\boldsymbol{\theta}, \mathbf{c})$ to above optimization problem is also a solution to (15). The dual as well

as the weighted updates now follow from the Lagrangian

$$\begin{aligned}\mathcal{L}(q, V, \eta_q, \eta_{\tilde{q}}, \lambda_q, \lambda_{\tilde{q}}) = \; & \mathbb{E}_{q(\boldsymbol{\theta},\mathbf{c})}\left[r\left(\boldsymbol{\theta},\mathbf{c}\right)\right] - \alpha D_{\mathrm{KL}}\left(q(\mathbf{c}) \parallel \mu(\mathbf{c})\right) \\ & + \eta_q\left(\epsilon - D_{\mathrm{KL}}\left(q(\boldsymbol{\theta},\mathbf{c}) \parallel p(\boldsymbol{\theta},\mathbf{c})\right)\right) + \lambda_q\left(1 - \int q(\boldsymbol{\theta},\mathbf{c})\,\mathrm{d}\mathbf{c}\,\mathrm{d}\boldsymbol{\theta}\right) \\ & + \eta_{\tilde{q}}\left(\epsilon - D_{\mathrm{KL}}\left(q(\mathbf{c}) \parallel p(\mathbf{c})\right)\right) + \lambda_{\tilde{q}}\left(1 - \int q(\mathbf{c})\,\mathrm{d}\mathbf{c}\right) \\ & + \int V(\mathbf{c})\left(\int q(\boldsymbol{\theta},\mathbf{c})\,\mathrm{d}\boldsymbol{\theta} - q(\mathbf{c})\right)\mathrm{d}\mathbf{c}. \end{aligned} \tag{32}$$

Note that we slightly abuse notation and overload the argument q in the definition of the Lagrangian. The update equations (16) and (17) follow from the two conditions $\frac{\partial L}{\partial q(\boldsymbol{\theta},\mathbf{c})} = 0$ and $\frac{\partial L}{\partial q(\mathbf{c})}\mathcal{L} = 0$. Inserting (16) and (17) into equation (32) then allows to derive the dual (18). We refer to Van Hoof et al. (2017) for detailed descriptions on the derivations in the non-contextual setting, which however generalize to the one investigated here.

Appendix C. Regularized Policy Updates

In order to enforce a gradual change in policy and context distribution not only during the computation of the weights via equations (16) and (17) but also during the actual inference of the new policy and context distribution, the default weighted linear regression and weighted maximum likelihood objectives need to be regularized. Given a data set of N weighted samples

$$D = \left\{(w_i^{\mathbf{x}}, w_i^{\mathbf{y}}, \mathbf{x}_i, \mathbf{y}_i) | i = 1, \ldots, N\right\},$$

with $\mathbf{x}_i \in \mathbb{R}^{d_{\mathbf{x}}}, \mathbf{y}_i \in \mathbb{R}^{d_{\mathbf{y}}}$, the task of fitting a joint-distribution

$$q(\mathbf{x}, \mathbf{y}) = q_{\mathbf{y}}(\mathbf{y}|\mathbf{x}) q_{\mathbf{x}}(\mathbf{x}) = \mathcal{N}(\mathbf{y}|\boldsymbol{A}\phi(\mathbf{x}), \boldsymbol{\Sigma}_{\mathbf{y}}) \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\mathbf{x}}, \boldsymbol{\Sigma}_{\mathbf{x}})$$

to D while limiting the change with regards to a reference distribution

$$p(\mathbf{x}, \mathbf{y}) = p_{\mathbf{y}}(\mathbf{y}|\mathbf{x}) p_{\mathbf{x}}(\mathbf{x}) = \mathcal{N}(\mathbf{y}|\tilde{\boldsymbol{A}}\phi(\mathbf{x}), \tilde{\boldsymbol{\Sigma}}_{\mathbf{y}}) \mathcal{N}(\mathbf{x}|\tilde{\boldsymbol{\mu}}_{\mathbf{x}}, \tilde{\boldsymbol{\Sigma}}_{\mathbf{x}}),$$

with feature function $\phi : \mathbb{R}^{d_x} \mapsto \mathbb{R}^o$, can be expressed as a constrained optimization problem

$$\begin{aligned} \max_{\boldsymbol{A}, \boldsymbol{\Sigma}_{\mathbf{y}}, \boldsymbol{\mu}_{\mathbf{x}}, \boldsymbol{\Sigma}_{\mathbf{x}}} & \sum_{i=1}^{N} (w_i^{\mathbf{x}} \log(q_{\mathbf{x}}(\mathbf{x}_i)) + w_i^{\mathbf{y}} \log(q_{\mathbf{y}}(\mathbf{y}_i|\mathbf{x}_i))) \\ \text{s.t. } & D_{\mathrm{KL}}\left(p \parallel q\right) \approx \frac{1}{N}\sum_{i=1}^{N} D_{\mathrm{KL}}\left(p_{\mathbf{y}}(\cdot|\mathbf{x}_i) \parallel q_{\mathbf{y}}(\cdot|\mathbf{x}_i)\right) + D_{\mathrm{KL}}\left(p_{\mathbf{x}} \parallel q_{\mathbf{x}}\right) \leq \epsilon. \end{aligned}$$

Note that we employ the reverse KL divergence in the constraint as this is the only form that allows for a closed form solution w.r.t. the parameters of the Gaussian distribution. Due to the unimodal nature of Gaussian distributions as well as the typically small value of ϵ this is a reasonable approximation. Since the distributions $p_{\mathbf{x}}$, $p_{\mathbf{y}}$, $q_{\mathbf{x}}$ and $q_{\mathbf{y}}$ are Gaussians,

the KL divergences can be expressed analytically. Setting the derivative of the Lagrangian with respect to the optimization variables to zero yields to following expressions of the optimization variables in terms of the multiplier η and the samples from D

$$\boldsymbol{A} = \left[\sum_{i=1}^{N} \left(w_i \mathbf{y}_i + \frac{\eta}{N} \tilde{\boldsymbol{A}} \phi(\mathbf{x}_i)\right) \phi(\mathbf{x}_i)^T\right] \left[\sum_{i=1}^{N} \left(w_i + \frac{\eta}{N}\right) \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\right]^{-1},$$

$$\boldsymbol{\Sigma}_{\mathbf{y}} = \frac{\sum_{i=1}^{N} w_i \Delta\mathbf{y}_i \Delta\mathbf{y}_i^T + \eta\tilde{\boldsymbol{\Sigma}}_{\mathbf{y}} + \frac{\eta}{N}\Delta\boldsymbol{A} \sum_{i=1}^{N} \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T \Delta\boldsymbol{A}^T}{\sum_{i=1}^{N} w_i + \eta},$$

$$\boldsymbol{\mu}_{\mathbf{x}} = \frac{\sum_{i=1}^{N} w_i \mathbf{x}_i + \eta\tilde{\boldsymbol{\mu}}_{\mathbf{x}}}{\sum_{i=1}^{N} w_i + \eta},$$

$$\boldsymbol{\Sigma}_{\mathbf{x}} = \frac{\sum_{i=1}^{N} w_i (\mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}})(\mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}})^T + \eta\left(\tilde{\boldsymbol{\Sigma}}_{\mathbf{x}} + (\boldsymbol{\mu}_{\mathbf{x}} - \tilde{\boldsymbol{\mu}}_{\mathbf{x}})(\boldsymbol{\mu}_{\mathbf{x}} - \tilde{\boldsymbol{\mu}}_{\mathbf{x}})^T\right)}{\sum_{i=1}^{N} w_i + \eta},$$

with $\Delta\mathbf{y}_i = \mathbf{y}_i - \boldsymbol{A}\phi(\mathbf{x}_i)$ and $\Delta\boldsymbol{A} = \boldsymbol{A} - \tilde{\boldsymbol{A}}$. Above equations yield a simple way of enforcing the KL bound on the joint distribution: Since η is zero if the constraint on the allowed KL divergence is not active, $\boldsymbol{A}$, $\boldsymbol{\Sigma}_{\mathbf{y}}$, $\boldsymbol{\mu}_{\mathbf{x}}$ and $\boldsymbol{\Sigma}_{\mathbf{x}}$ can be first computed with $\eta = 0$ and only if the allowed KL divergence is exceeded, η needs to be found by searching the root of

$$f(\eta) = \epsilon - \frac{1}{N}\sum_{i=1}^{N} D_{\mathrm{KL}}\left(p_{\mathbf{y}}(\cdot|\mathbf{x}_i) \parallel q_{\mathbf{y}}(\cdot|\mathbf{x}_i)\right) + D_{\mathrm{KL}}\left(p_{\mathbf{x}} \parallel q_{\mathbf{x}}\right),$$

where $q_{\mathbf{y}}$ and $q_{\mathbf{x}}$ are expressed as given by above formulas and hence implicitly depend on η. As this is a one-dimensional root finding problem, simple algorithms can be used for this task.

Appendix D. Experimental Details

This section is composed of further details on the experiments in sections 6 and 7, which were left out in the main paper to improve readability. The details are split between the episodic- and step-based scenarios as well as the individual experiments conducted in them. To conduct the experiments, we use the implementation of ALP-GMM, GoalGAN and SAGG-RIAC provided in the repositories accompanying the papers from Florensa et al. (2018) and Portelas et al. (2019) as well as the CMA-ES implementation from Hansen et al. (2019). The employed hyperparameters are discussed in the corresponding sections.
Conducting the experiments with SPRL and SPDL, we found that restricting the standard deviation of the context distribution $p(\mathbf{c}|\boldsymbol{\nu})$ to stay above a certain lower bound $\boldsymbol{\sigma}_{\mathrm{LB}}$ helps to stabilize learning when generating curricula for narrow target distributions. This is because the Gaussian distributions have a tendency to quickly reduce the variance of the sampling distribution in this case. In combination with the KL divergence constraint on subsequent

context distributions, this slows down progression towards the target distribution. Although we could enforce aforementioned lower bound via constraints on the distribution $p(\mathbf{c}|\boldsymbol{\nu})$, we simply clip the standard deviation until the KL divergence w.r.t. the target distribution $\mu(\mathbf{c})$ falls below a certain threshold $D_{\mathrm{KL}_{\mathrm{LB}}}$. This threshold was chosen such that the distribution with the clipped standard deviation roughly "contains" the mean of target distribution within its standard deviation interval. The specific values of $D_{\mathrm{KL}_{\mathrm{LB}}}$ and $\boldsymbol{\sigma}_{\mathrm{LB}}$ are listed for the individual experiments.

D.1 Episodic Setting

For the visualization of the success rate as well as the computation of the success indicator for the GoalGAN algorithm, the following definition is used: An experiment is considered successful, if the distance between final- and desired state ($\mathbf{s}_f$ and $\mathbf{s}_g$) is less than a given threshold τ

$$\text{Success}\left(\boldsymbol{\theta}, \mathbf{c}\right) = \begin{cases} 1, \text{ if } \left\|\mathbf{s}_f\left(\boldsymbol{\theta}\right) - \mathbf{s}_g\left(\mathbf{c}\right)\right\|_2 < \tau, \\ 0, \text{ else.} \end{cases}$$

For the Gate and reacher environment, the threshold is fixed to 0.05, while for the ball-in-a-cup environment, the threshold depends on the scale of the cup and the goal is set to be the center of the bottom plate of the cup.
The policies are chosen to be conditional Gaussian distributions $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{A}\phi(\mathbf{c}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$, where $\phi(\mathbf{c})$ is a feature function. SPRL and C-REPS both use linear policy features in all environments.
In the reacher and the ball-in-a-cup environment, the parameters $\boldsymbol{\theta}$ encode a feed-forward policy by weighting several Gaussian basis functions over time

$$\boldsymbol{u}_i\left(\boldsymbol{\theta}\right) = \boldsymbol{\theta}^T \boldsymbol{\psi}\left(t_i\right), \quad \boldsymbol{\psi}_j\left(t_i\right) = \frac{b_j\left(t_i\right)}{\sum_{l=1}^{L} b_l\left(t_i\right)}, \quad b_j\left(t_i\right) = \exp\left(\frac{\left(t_i - c_j\right)^2}{2L}\right),$$

where the centers c_j and length L of the basis functions are chosen individually for the experiments. With that, the policy represents a Probabilistic Movement Primitive (Paraschos et al., 2013), whose mean and covariance matrix are progressively shaped by the learning algorithm to encode movements with high reward.

	ϵ	n_{samples}	Buffer Size	ζ	K_α	$\boldsymbol{\sigma}_{\mathrm{LB}}$	$D_{\mathrm{KL}_{\mathrm{LB}}}$
Gate "Global"	0.25	100	10	0.002	140	-	-
Gate "Precision"	0.4	100	10	0.02	140	-	-
Reacher	0.5	50	10	0.15	90	[0.005 0.005]	20
Ball-in-a-cup	0.35	16	5	3.0	15	0.1	200

Table 3: Important parameters of SPRL and C-REPS in the conducted experiments. The meaning of the symbols correspond to those presented in the algorithm from the main text and introduced in this appendix.

	δ_{NOISE}	$n_{\text{ROLLOUT}_{\text{GG}}}$	n_{GOALS}	n_{HIST}
Gate "Global"	0.05	5	100	500
Gate "Precision"	0.05	5	100	200
Reacher	0.1	5	80	300
Ball-in-a-cup	0.05	3	50	120

Table 4: Important parameters of GoalGAN and SAGG-RIAC in the conducted experiments. The meaning of the symbols correspond to those introduced in this appendix.

In order to increase the robustness of SPRL and C-REPS while reducing the sample complexity, an experience buffer storing samples of recent iterations is used. The size of this buffer dictates the number of past iterations, whose samples are kept. Hence, in every iteration, C-REPS and SPRL work with $N_{\text{SAMPLES}} \times$ BUFFER SIZE samples, from which only N_{SAMPLES} are generated by the policy of the current iteration.
As the employed CMA-ES implementation only allows to specify one initial variance for all dimensions of the search distribution, this variance is set to the maximum of the variances contained in the initial covariance matrices used by SPRL and C-REPS.
For the GoalGAN algorithm, the percentage of samples that are drawn from the buffer containing already solved tasks is fixed to 20%. The noise added to the samples of the GAN δ_{NOISE} and the number of iterations that pass between the training of the GAN $n_{\text{ROLLOUT}_{\text{GG}}}$ are chosen individually for the experiments.
The SAGG-RIAC algorithm requires, besides the probabilities for the sampling modes which are kept as in the original paper, two hyperparameters to be chosen: The maximum number of samples to keep in each region n_{GOALS} as well as the maximum number of recent samples for the competence computation n_{HIST}.
Tables 3 and 4 show the aforementioned hyperparameters of C-REPS, SPRL, GoalGAN and SAGG-RIAC for the different environments.

D.1.1 Point-Mass Experiment

The linear system that describes the behavior of the point-mass is given by

$$\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = \begin{bmatrix}5\\ -1\end{bmatrix} + \boldsymbol{u} + \boldsymbol{\delta}, \quad \boldsymbol{\delta} \sim \mathcal{N}\left(\mathbf{0}, 2.5 \times 10^{-3}\boldsymbol{I}\right).$$

The point-mass is controlled by two linear controllers

$$\mathrm{C}_i\left(x, y\right) = \boldsymbol{K}_i \begin{bmatrix}x_i - x\\ y_i - y\end{bmatrix} + \boldsymbol{k}_i,\ i \in [1, 2],\quad \boldsymbol{K}_i \in \mathbb{R}^{2\times 2},\ \boldsymbol{k}_i \in \mathbb{R}^2,\ x_i, y_i \in \mathbb{R},$$

where x is the x-position of the point-mass and y its position on the y-axis. The episode reward exponentially decays with the final distance to the goal. In initial iterations of the algorithm, the sampled controller parameters sometimes make the control law unstable, leading to very large penalties due to large actions and hence to numerical instabilities in

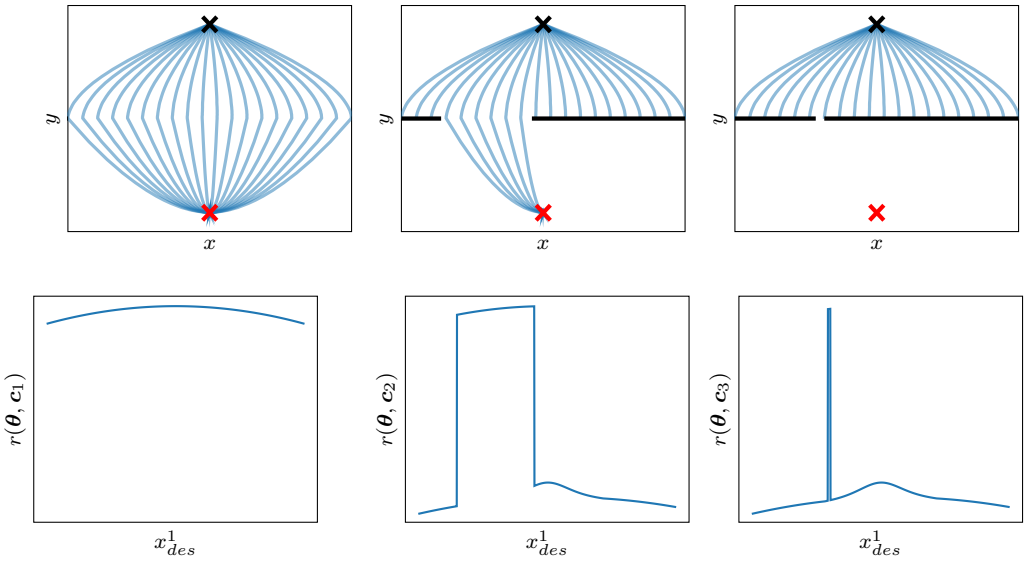


Figure 11: The columns show visualizations of the point-mass trajectories (upper plots) as well as the obtained rewards (lower plots) in the point-mass task, when the desired position of the first controller is varied while all other parameters are kept fixed such that a stable control law is obtained. In every column, the gate is positioned at $x = 4.0$ while the size of it varies from 20 (left), over 3 (middle) to 0.1 (right).

SPRL and C-REPS because of very large negative rewards. Because of this, the reward is clipped to always be above 0.
Table 3 shows that a large number of samples per iteration for both the "global" and "precision" setting are used. This is purposefully done to keep the influence of the sample size on the algorithm performance as low as possible, as both of these settings serve as a first conceptual benchmark of our algorithm.
Figure 11 helps in understanding, why SPRL drastically improves upon C-REPS especially in the "precision" setting, even with this large amount of samples. For narrow gates, the reward function has a local maximum which tends to attract both C-REPS and CMA-ES, as the chance of sampling a reward close to the true maximum is very unlikely. By first training on contexts in which the global maximum is more likely to be observed and only gradually moving towards the desired contexts, SPRL avoids this sub-optimal solution.

D.1.2 Reacher Experiment

In the reacher experiment, the ProMP encoded by the policy π has 20 basis functions of width $L = 0.03$. The centers are evenly spread in the interval $[-0.2, 1.2]$ and the time interval of the movement is normalized to lie in the interval $[0, 1]$ when computing the activations of the basis functions. Since the robot can only move within the xy-plane, $\boldsymbol{\theta}$ is a 40-dimensional vector. As in the previous experiment, the episode reward decays

exponentially with the final distance to the goal. As we can see in Table 3, the number of samples in each iteration was decreased to 50, which in combination with the increased dimensionality of $\boldsymbol{\theta}$ makes the task more challenging.
As in the step-based setting, the PPO results are obtained using the version from the `Stable Baselines` library (Hill et al., 2018). A step-based version of the reacher experiment is used, in which the reward function is given by

$$r(\boldsymbol{s}, \boldsymbol{a}) = \exp\left(-2.5\sqrt{(x - x_g)^2 + (y - y_g)^2}\right),$$

where $\boldsymbol{s} = (x\ \dot{x}\ y\ \dot{y})$ is the position and velocity of the end-effector, $\boldsymbol{a} = (a_x\ a_y)$ the desired displacement of the end-effector (just as in the regular reacher task from the OpenAI Gym simulation environment) and x_g and y_g is the $x-$ and $y-$ position of the goal. When an obstacle is touched, the agent is reset to the initial position. This setup led to the best performance of PPO, while resembling the structure of the episodic learning task used by the other algorithms (a version in which the episode ends as soon as an obstacle is touched led to a lower performance of PPO).
To ensure that the poor performance of PPO is not caused by an inadequate choice of hyperparameters, PPO was run on an easy version of the task in which the two obstacle sizes were set to 0.01, where it encountered no problems in solving the task.
Every iteration of PPO uses 3600 environment steps, which corresponds to 24 trajectory executions in the episodic setting. PPO uses an entropy coefficient of 10^{-3}, $\gamma = 0.999$ and $\lambda = 1$. The neural network that learns the value function as well as the policy has two dense hidden layers with 164 neurons and tanh activation functions. The number of minibatches is set to 5 while the number of optimization epochs is set to 15. The standard deviation in each action dimension is initialized to 1, giving the algorithm enough initial variance, as the actions are clipped to the interval $[-1, 1]$ before being applied to the robot.

D.1.3 Ball-in-a-Cup Experiment

For the ball-in-a-cup environment, the 9 basis functions of the ProMP are spread over the interval $[-0.01, 1.01]$ and have width $L = 0.0035$. Again, the time interval of the movement is normalized to lie in the interval $[0, 1]$ when computing the basis function activations. The ProMP encodes the offset of the desired position from the initial position. By setting the first and last two basis functions to 0 in each of the three dimensions, the movement always starts in the initial position and returns to it after the movement execution. All in all, θ is a 15-dimensional vector. The reward function is defined as

$$r(\boldsymbol{\theta}, \mathbf{c}) = \begin{cases} 1 - 0.07\boldsymbol{\theta}^T\boldsymbol{\theta} & , \text{ if successful} \\ 0 & , \text{ else} \end{cases}.$$

This encodes a preference over movements that deviate as little as possible from the initial position while still solving the task.
Looking back at Table 3, the value of ζ stands out, as it is significantly higher than in the other experiments. We suppose that such a large value of ζ is needed because of the shape of the reward function, which creates a large drop in reward if the policy is sub-optimal. Because of this, the incentive required to encourage the algorithm to shift probability mass

	K_α	ζ	K_{OFFSET}	V_{LB}	n_{STEP}	$\boldsymbol{\sigma}_{\text{LB}}$	$D_{\text{KL}_{LB}}$
Point-Mass (TRPO)	20	1.6	5	3.5	2048	[0.2 0.1875 0.1]	8000
Point-Mass (PPO)	10	1.6	5	3.5	2048	[0.2 0.1875 0.1]	8000
Point-Mass (SAC)	25	1.1	5	3.5	2048	[0.2 0.1875 0.1]	8000
Ant (PPO)	15	1.6	10	600	81920	[1 0.5]	11000
Ball-Catching (TRPO)	70	0.4	5	42.5	5000	-	-
Ball-Catching* (TRPO)	0	0.425	5	42.5	5000	-	-
Ball-Catching (PPO)	50	0.45	5	42.5	5000	-	-
Ball-Catching* (PPO)	0	0.45	5	42.5	5000	-	-
Ball-Catching (SAC)	60	0.6	5	25	5000	-	-
Ball-Catching* (SAC)	0	0.6	5	25	5000	-	-

Table 5: 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.

towards contexts in which the current policy is sub-optimal needs to be significantly higher than in the other experiments.
After learning the movements in simulation, the successful runs were executed on the real robot. Due to simulation bias, just replaying the trajectories did not work satisfyingly. At this stage, we could have increased the variance of the movement primitive and re-trained on the real robot. As sim-to-real transfer is, however, not the focus of this paper, we decided to manually adjust the execution speed of the movement primitive by a few percent, which yielded the desired result.

D.2 Step-Based Setting

The parameters of SPDL for different environments and RL algorithms are shown in Table 5. Opposed to the sketched algorithm in the main paper, we specify the number of steps n_{STEP} in the environment between context distribution updates instead of the number of trajectory rollouts. The additional parameter K_{OFFSET} describes the number of RL algorithm iterations that take place before SPDL is allowed the change the context distribution. We used this in order to improve the estimate regarding task difficulty, as for completely random policies, task difficulty is not as apparent as for slightly more structured ones. This procedure corresponds to providing parameters of a minimally 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 point-mass 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.
Since the step-based algorithm makes use of the value function estimated by the individual RL algorithms, particular regularizations of RL algorithms can affect the curriculum. SAC, for example, estimates a "biased" value function due to the employed entropy regularization. This bias caused problems for our algorithm when working with the α-heuristic based on V_{LB}. Because of this, we simply replace the value estimates for the contexts by their sample

return when working with SAC and V_{LB}. This is an easy way to obtain an unbiased, yet noisier estimate of the value of a context. Furthermore, the general advantage estimation (GAE) employed by TRPO and PPO can introduce bias in the value function estimates as well. For the ant environment, we realized that this bias is particularly large due to the long time horizons. Consequently, we again made use of the sample returns to estimate the value functions for the sampled contexts. In all other cases and environments, we used the value functions estimated by the RL algorithms.
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

$$(p_{\text{RAND}}, n_{\text{ROLLOUT}}, s_{\text{BUFFER}}) \in \{0.1, 0.2, 0.3\} \times \{25, 50, 100, 200\} \times \{500, 1000, 2000\}.$$

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

$$(\delta_{\text{NOISE}}, n_{\text{ROLLOUT}}, p_{\text{SUCCESS}}) \in \{0.025, 0.05, 0.1\} \times \{25, 50, 100, 200\} \times \{0.1, 0.2, 0.3\}.$$

The results of the hyperparameter optimization for GoalGAN and ALP-GMM are shown in Table 6.
Since for all environments, both initial- and target distribution are Gaussians with independent noise in each dimension, we specify them in Table 7 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.
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.

D.2.1 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 $\mathbf{c} = [p_g\ w_g\ \mu_k] \in [-4, 4] \times [0.5, 8] \times [0, 4] \subset \mathbb{R}^3$. 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

	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

Table 6: 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.

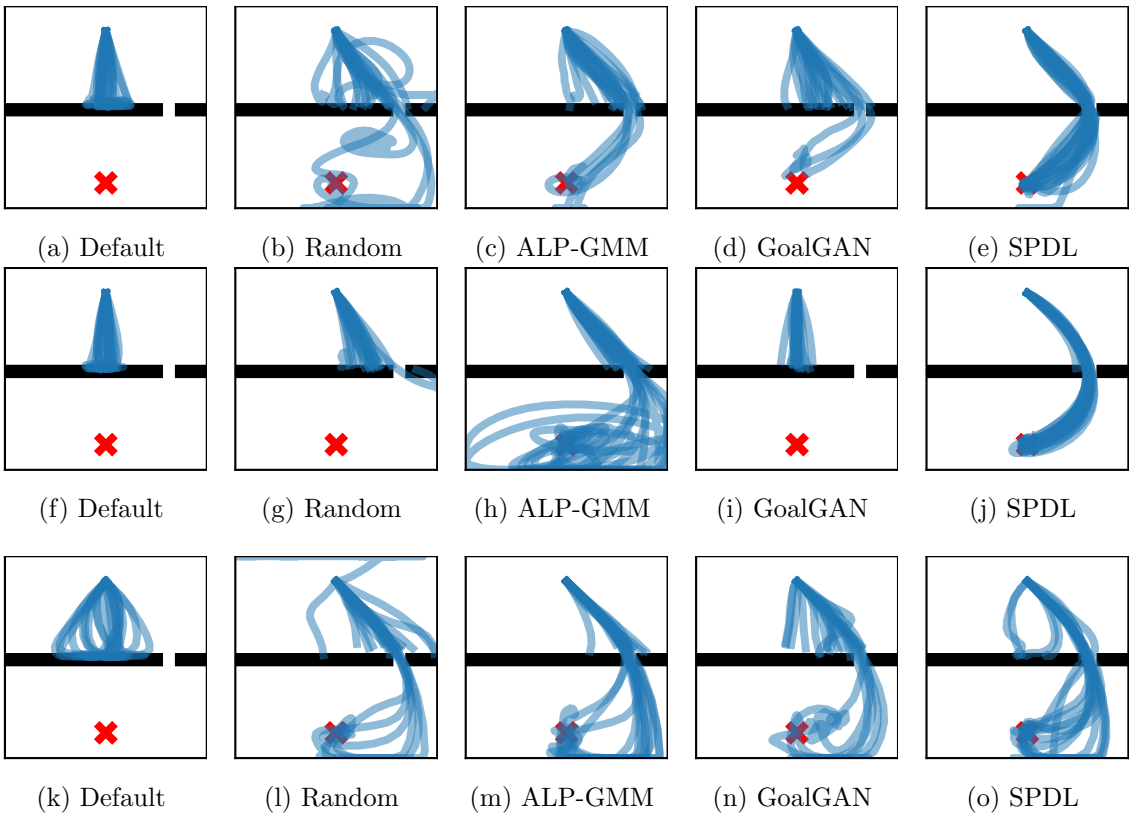
	$\boldsymbol{\mu}_{\text{INIT}}$	$\boldsymbol{\delta}_{\text{INIT}}$	$\boldsymbol{\mu}_{\text{TARGET}}$	$\boldsymbol{\delta}_{\text{TARGET}}$
Point-Mass	[0 4.25 2]	[2 1.875 1]	[2.5 0.5 0]	[0.004 0.00375 0.002]
Ant	[0 8]	[3.2 1.6]	[−8 3]	[0.01 0.005]
Ball-Catching	[0.68 0.9 0.85]	[0.03 0.03 0.3]	[1.06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

Table 7: Mean and standard deviation of target and initial distributions per environment.

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 two hidden layers with 64 neurons and tanh activations. For TRPO and PPO, we take 2048 steps in the environment between policy updates.
For TRPO we set the GAE parameter $\lambda = 0.99$, 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, starting learning after 500 steps. We use the soft Q-Updates and update the policy every 5 environment steps. All other parameters were left at their implementation defaults.
For SPRL, we use $K_\alpha = 40$, $K_{\text{OFFSET}} = 0$, $\zeta = 2.0$ 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 5). 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 to approximate the value function.



(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 12: 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.

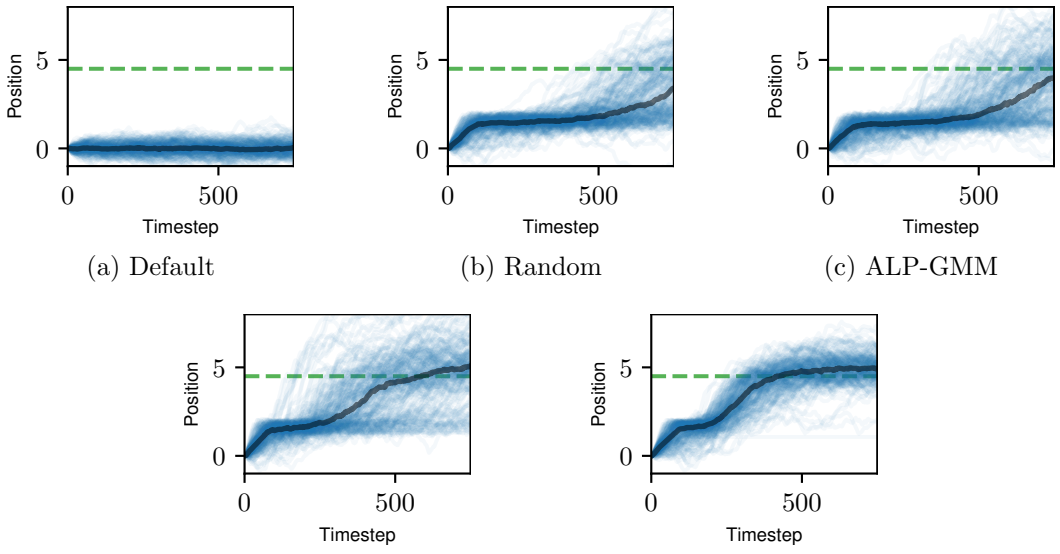
For the allowed KL divergence, we observed best results when using $\epsilon = 0.5$ for the weight computation of the samples, but using a lower value of $\epsilon = 0.2$ when fitting the parametric policy to these samples. We suppose that the higher value of ϵ during weight computation counteracts the effect of the buffer containing policy samples from earlier iterations.
Looking at Figure 12, we can see that depending on the learning algorithm, ALP-GMM, GoalGAN and a random curriculum 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 GoalGAN compared to directly learning on the target task, as a crash prevents the agent from accumulating positive rewards over time. SPDL learned more reliable and directed policies across all learning algorithms.

D.2.2 Ant Environment

As mentioned in the main paper, we simulate the ant using the Isaac Gym simulator (Nvidia, 2019). 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 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 $\mathbf{c} = [p_g \ w_g] \in [-10, 10] \times [3, 13] \subset \mathbb{R}^2$ 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 (Brockman et al., 2016). 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



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

(d) GoalGAN (e) SPDL

Figure 13: 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.

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 13, 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.

D.2.3 Ball-Catching Environment

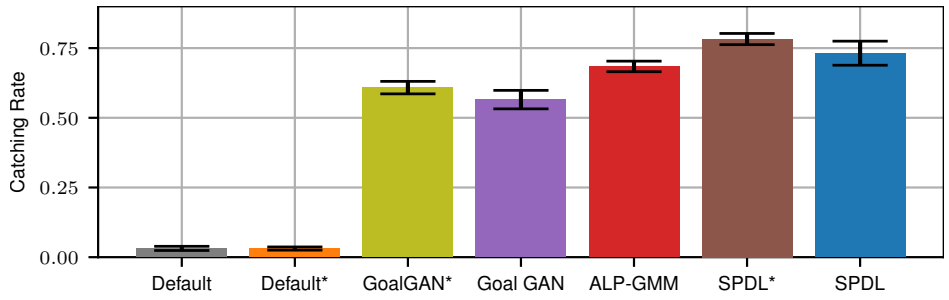
In the final environment, the robot is controlled in joint space via the desired position for 5 of the 7 joints. We only control a subspace of all available joints, since it is not necessary for the robot to leave the "catching" plane (defined by $x = 0$) that is intersected by each ball. The actions $\boldsymbol{a} \in \mathbb{R}^5$ 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,

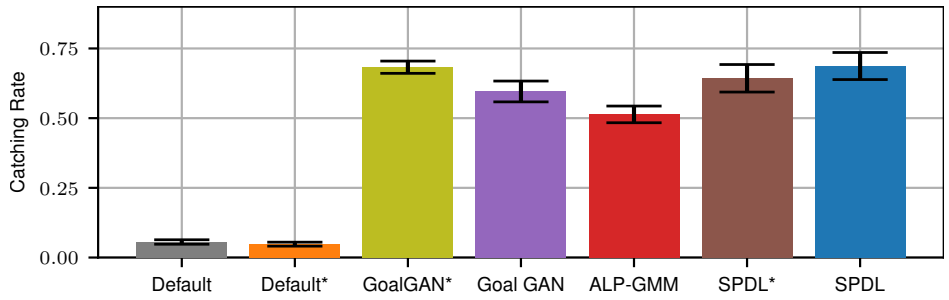
$$r(\boldsymbol{s}, \boldsymbol{a}) = 0.275 - 0.005\|\boldsymbol{a}\|_2^2 + \begin{cases} 50 + 25(\boldsymbol{n}_s \cdot \boldsymbol{v}_b)^5, \text{ if ball catched} \\ 0, \text{ else} \end{cases},$$

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.

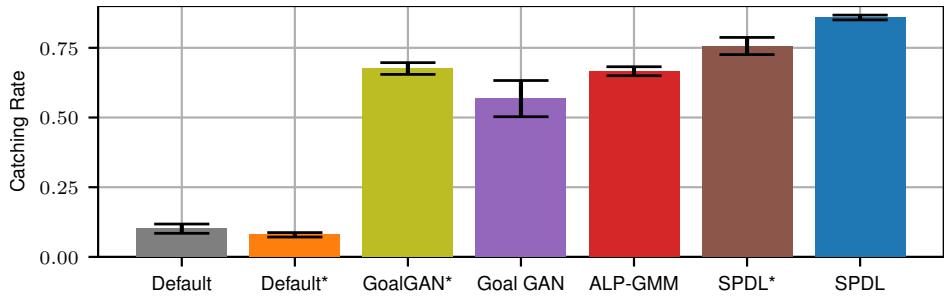
$$\boldsymbol{p}_{\text{des}} = [0 \quad -r\cos(\phi) \quad 0.75 + r\sin(\phi)].$$



(a) SAC



(b) TRPO



(c) PPO

Figure 14: 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.

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 $t = 0.5 + 0.05 d_x$. 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 14 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. 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.

References

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations (ICLR)*, 2018.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23 (2):279–303, 1996.

Adrien Baranes and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration for active motor learning in robots: A case study. In *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Neural Information Processing Systems (NeurIPS)*, 2016.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. Safe controller optimization for quadrotors with gaussian processes. In *International Conference on Robotics and Automation (ICRA)*, 2016.

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*, 36(2):125–150, 2005.

Josh Bongard and Hod Lipson. Once more unto the breach: Co-evolving a robot and its simulator. In *Conference on Artificial Life (ALIFE)*, 2004.

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

Andrew R Conn, Katya Scheinberg, and Luis N Vicente. *Introduction to Derivative-Free Optimization*. SIAM, 2009.

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

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

Tom Erez and William D Smart. What does shaping mean for computational reinforcement learning? In *International Conference on Development and Learning (ICDL)*, 2008.

Hehe Fan, Liang Zheng, Chenggang Yan, and Yi Yang. Unsupervised person re-identification: Clustering and fine-tuning. *Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 14(4):1–18, 2018.

Matthew Fellows, Anuj Mahajan, Tim GJ Rudner, and Shimon Whiteson. Virel: A variational inference framework for reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2017.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning (ICML)*, 2018.

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

Andrew Gelman and Xiao-Li Meng. Simulating normalizing constants: From importance sampling to bridge sampling to path sampling. *Statistical Science*, 13(2):163–185, 1998.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8(5-6): 359–483, 2015.

Matthew M. Graham and Amos J. Storkey. Continuously tempered hamiltonian monte carlo. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1):1–18, 2003.

Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2010.

Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. CMA-ES/pycma on Github, February 2019. URL `https://github.com/CMA-ES/pycma`.

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.

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.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Neural Information Processing Systems (NeurIPS)*, 2016.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations (ICLR)*, 2017.

Lu Jiang, Deyu Meng, Teruko Mitamura, and Alexander G Hauptmann. Easy samples first: Self-paced reranking for zero-example multimedia search. In *ACM International Conference on Multimedia (MM)*, 2014a.

Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. Self-paced learning with diversity. In *Neural Information Processing Systems (NeurIPS)*, 2014b.

Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G Hauptmann. Self-paced curriculum learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

Jens Kober and Jan Peters. Policy search for motor primitives in robotics. In *Neural Information Processing Systems (NeurIPS)*, 2009.

M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Neural Information Processing Systems (NeurIPS)*, 2010.

Andras Gabor Kupcsik, Marc Peter Deisenroth, Jan Peters, and Gerhard Neumann. Data-efficient generalization of robot skills with contextual policy search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2013.

Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

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

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research (JMLR)*, 17(1):1334–1373, 2016.

Chang Liu, Jingwei Zhuo, Pengyu Cheng, Ruiyi Zhang, Jun Zhu, and Lawrence Carin. Understanding and accelerating particle-based variational inference. In *International Conference on Machine Learning (ICML)*, 2019.

Rui Luo, Jianhong Wang, Yaodong Yang, Wang Jun, and Zhanxing Zhu. Thermostat-assisted continuously-tempered hamiltonian monte carlo for bayesian learning. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Stephan Mandt, James McInerney, Farhan Abrol, Rajesh Ranganath, and David Blei. Variational tempering. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

Enzo Marinari and Giorgio Parisi. Simulated tempering: A new Monte Carlo scheme. *Europhysics Letters*, 19:451–458, 1992.

Deyu Meng, Qian Zhao, and Lu Jiang. A theoretical understanding of self-paced learning. *Information Sciences*, 414:319–328, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Aditya Modi, Nan Jiang, Satinder Singh, and Ambuj Tewari. Markov decision processes with continuous side information. In *International Conference on Algorithmic Learning Theory (ALT)*, 2018.

Sanmit Narvekar and Peter Stone. Learning curriculum policies for reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2019.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research (JMLR)*, 21(181):1–50, 2020.

Gerhard Neumann. Variational inference for policy search in changing situations. In *International Conference on Machine Learning (ICML)*, 2011.

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

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.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Transactions on Knowledge and Data Engineering (TKDE)*, 22(10):1345–1359, 2009.

Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Probabilistic movement primitives. In *Neural Information Processing Systems (NeurIPS)*, 2013.

Simone Parisi, Hany Abdulsamad, Alexandros Paraschos, Christian Daniel, and Jan Peters. Reinforcement learning vs human programming in tetherball robot games. In *International Conference on Intelligent Robots and Systems (IROS)*, 2015.

Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

Kai Ploeger, Michael Lutter, and Jan Peters. High acceleration reinforcement learning for real-world juggling with binary rewards. In *Conference on Robot Learning (CoRL)*, 2020.

Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning (CoRL)*, 2019.

Simon JD Prince. *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, 2012.

Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

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.

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. In *International Conference on Machine Learning (ICML)*, 2018.

Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*, pages 261–280. Springer, 2006.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning (ICML)*, 2015.

Jürgen Schmidhuber. Curious model-building control systems. In *International Joint Conference on Neural Networks (IJCNN)*, 1991.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.

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

Matthias Schultheis, Boris Belousov, Hany Abdulsamad, and Jan Peters. Receding horizon curiosity. In *Conference on Robot Learning (CoRL)*, 2020.

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, 2017.

Burrhus Frederic Skinner. *The Behavior of Organisms: An Experimental Analysis*. Appleton-Century, 1938.

James S Supancic and Deva Ramanan. Self-paced learning for long-term tracking. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2379–2386, 2013.

Richard S Sutton and Andrew G Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research (JMLR)*, 10(7), 2009.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *International Conference on Machine Learning (ICML)*, 2006.

Naonori Ueda and Ryohei Nakano. Deterministic annealing variant of the em algorithm. In *Neural Information Processing Systems (NeurIPS)*, 1995.

Herke Van Hoof, Gerhard Neumann, and Jan Peters. Non-parametric policy search with limited information loss. *Journal of Machine Learning Research (JMLR)*, 18(73):1–46, 2017.

Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*, pages 7–15. Springer, 1987.

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.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Poet: open-ended coevolution of environments and their optimized solutions. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 142–151, 2019.

Andre Wibisono. Sampling as optimization in the space of measures: The langevin dynamics as a composite optimization problem. In *Conference on Learning Theory (COLT)*, 2018.

Jan Wöhlke, Felix Schmitt, and Herke van Hoof. A performance-based start state curriculum framework for reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1503–1511, 2020.

Cun-Hui Zhang. Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, 38(2):894–942, 2010.

Tong Zhang. Multi-stage convex relaxation for learning with sparse regularization. In *Neural Information Processing Systems (NeurIPS)*, pages 1929–1936, 2008.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2008.