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

Pascal Klink
Intelligent Autonomous Systems, TU Darmstadt, Germany

Hany Abdulsamad
Intelligent Autonomous Systems, TU Darmstadt, Germany

Boris Belousov
Intelligent Autonomous Systems, TU Darmstadt, Germany

Carlo D’Eramo
Intelligent Autonomous Systems, TU Darmstadt, Germany

Jan Peters
Intelligent Autonomous Systems, TU Darmstadt, Germany

Joni Pajarinen
Intelligent Autonomous Systems, TU Darmstadt, Germany

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, RL-as-Inference, Tempered 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), initially investigated by Bengio et al. (2009) for supervised learning problems. By now, applications have expanded to RL problems, where the aim is to design task sequences that maximally benefit the learning progress of an RL agent (?).

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 both 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-minimization 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 which 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 8). We experimentally evaluate the 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 (Erez and Smart, 2008). For supervised learning (SL), this principle was given the name curriculum learning by Bengio et al. (2009). The name has by now also been established in the RL community, where a variety of algorithms have been proposed that aim to generate curricula that maximally benefit the learner.

A driving principle behind curriculum learning for RL 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 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 is commonly assumed. There are multiple names for this additional variable, sometimes being referred to as goals (Schaul et al., 2015), sometimes as contexts (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 clearly an interesting endeavour. We now continue to highlight some CRL algorithms and refer to the survey by ? 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. To the best of our knowledge, only our work and that of Wöhlke et al. (2020), which particularly focuses on a curriculum over initial environment states, fall into this category.

Our approach to curriculum generation builds upon the idea of self-paced learning (SPL), initially proposed by Kumar et al. (2010) for SL 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-minimization algorithm applied to a regularized objective function. Despite its well-understood theoretical standing and empirical success in SL tasks, SPL has only been applied in a limited way to RL problems, restricting its use to the regression of a value function from an experience buffer (Ren et al., 2018). We, however, will make use of SPL to adaptively select training tasks during learning of the agent.

Furthermore, we will connect 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). For RL, it motivates regularization techniques such as the concept of maximum- or relative entropy (Ziebart et al., 2008; Peters et al., 2010; Haarnoja et al., 2018) and stimulates the development of new, and interpretation of, existing algorithms from a common view (Abdolmaleki et al., 2018; Fellows et al., 2019).

3. Preliminaries

This section introduces 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), augments the optimization of a function approximator with parameters $\omega \in \mathbb{R}^{d\omega}$ w.r.t. a given dataset $D = \{(x_i, y_i) \mid x_i \in \mathbb{R}^{d_x}, y_i \in \mathbb{R}, i \in [1, N]\}$ with additional variables $\nu \in [0, 1]^N$ as well as a “self-paced regularizer” $f(\alpha, \nu_i)$

$$\nu^*, \omega^* = \arg\min_{\nu, \omega} r(\omega) + \sum_{i=1}^{N} (\nu_i f(x_i, y_i, \omega) + f(\alpha, \nu_i)), \quad \alpha > 0. \quad (1)$$

The term $r(\omega)$ represents potentially employed regularization of the objective w.r.t. $\omega$ and $f(x_i, y_i, \omega)$ represents the model error for sample $(x_i, y_i)$. The motivation for this principle as well as its name are best explained by investigating the solution $\nu^*(\alpha, \omega)$ of the augmented optimization problem when only optimizing it w.r.t. $\nu$ while keeping $\alpha$ and $\omega$ fixed. For the self-paced function $f_{\text{Bin}}(\alpha, \nu_i) = -\alpha \nu_i$ initially proposed by Kumar et al.
We see that \( \nu^* \) focuses on examples on which the approximator under the current parameters performs better than a chosen threshold \( \alpha \). By continuously increasing \( \alpha \) and updating \( \nu \) and \( \omega \) in a block-coordinate manner, SPL creates a curriculum consisting of increasingly “hard” training examples w.r.t. the current approximator. Jiang et al. (2015) formalized this idea via three axioms that a self-paced regularizer needs to fulfill.

Defining \( \nu^*_i(\alpha, l) = \arg\min_{\nu_i} \nu_i \nu_i l + f(\alpha, \nu_i) \), these axioms are

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

This definition was also used in the interpretation of SPL as a majorize-minimization (MM) algorithm to show that for particular choices of self-paced regularizers, well-known regularization terms can be recovered (Meng et al., 2017). More precisely, Meng et al. (2017) show that the SPL scheme of alternatingly optimizing (1) w.r.t. \( \omega \) and \( \nu \) implicitly optimizes the regularized objective

\[
\min_{\omega} r(\omega) + \sum_{i=1}^{N} F_\alpha(f(x_i, y_i, \omega)), \quad F_\alpha(f(x_i, y_i, \omega)) = \int_0^{f(x_i, y_i, \omega)} \nu^*_i(\alpha, l) dl.
\] (3)

Meng et al. (2017) could prove this by making use of the monotonically decreasing behavior of \( \nu_i(\alpha, l) \) w.r.t. \( l \). Furthermore, they showed that, depending on the chosen self-paced regularizer, \( F_\alpha(f(x_i, y_i, \omega)) \) corresponds to well-known regularization techniques such as the capped-norm based penalty or the minimax concave plus penalty, which are known to e.g. reduce the sensibility to outliers when fitting a function approximator to data. This formulation yields an appealing interpretation of curriculum learning performing an implicit regularization of the objective function. The focus of this paper is the application of this insight 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 \( M = (S, A, p, r, p_0) \) that defines an environment with states \( s \in S \), actions \( a \in A \), transition probabilities \( p(s'|s, a) \), reward function \( r : S \times A \mapsto \mathbb{R} \) and initial state distribution \( p_0(s) \). Typically \( S \) and \( A \) are discrete spaces or subsets of \( \mathbb{R}^n \). RL encompasses approaches that maximize a \( \gamma \)-discounted performance measure

\[
\max_{\omega} J(\omega) = \max_{\omega} \mathbb{E}_{p_0(s_0), p(s_{i+1}|s_i, a_i), \pi(a_i|s_i, \omega)} \left[ \sum_{i=0}^{\infty} \gamma^i r(s_i, a_i) \right]
\] (4)
by finding optimal parameters $\omega$ for the policy $\pi(a|s, \omega)$ through interaction with the environment. A key ingredient to many RL algorithms is the value function

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

which encodes the long-term expected discounted reward of following policy $\pi(\cdot|\cdot, \omega)$ from state $s$. The value function (or an estimate of it) is related to the RL objective by $J(\omega) = \mathbb{E}_{p_0(s)} \left[ V_\omega(s) \right]$. 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 $c \in C \subseteq \mathbb{R}^m$, i.e. $\mathcal{M}(c) = (S, A, p_c, r_c, p_{0,c})$. By conditioning the agents behavior on this context, i.e. $\pi(a|s, c, \omega)$, and introducing a distribution over the contextual parameter $\mu(c)$ we end up with a contextual RL objective

$$\max_{\omega} J(\omega, \mu) = \max_{\omega} \mathbb{E}_{\mu(c)} \left[ J(\omega, c) \right] = \max_{\omega} \mathbb{E}_{\mu(c), p_0,c(s)} \left[ V_\omega(s, c) \right].$$

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

$$V_\omega(s, c) = \mathbb{E}_{\pi(a|s,c,\omega)} \left[ r_c(s, a) + \gamma \mathbb{E}_{p_c(s'|s, a)} \left[ V_\omega(s', c) \right] \right].$$

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 nonlinear objectives (4) and (6), which may or may not encode behavior that is able to solve the task (or the distribution of them). 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 $\mathcal{M}(c)$, in which learning can robustly take place despite aforementioned approximations, e.g. since there is only one solution to objective (4). Furthermore, consider a second task $\mathcal{M}(c')$ which now admits multiple solutions with different expected reward. If the solution of $\mathcal{M}(c)$ lies within the basin of attraction of the optimal solution to $\mathcal{M}(c')$, first learning in $c$ and afterwards in $c'$ promises to stabilize learning towards the optimal solution. 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 $p_i(c)$. This sequence should converge to a desired target distribution $\mu(c)$, i.e. $\lim_{i \to \infty} p_i(c) = \mu(c)$.

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 $\|c - c'\|$ implies a certain similarity between the tasks $\mathcal{M}(c)$ and $\mathcal{M}(c')$. Note that the imprecision of this formulation is not by accident but is rather an acknowledgement that the question of similarity between MDPs is a complicated topic on its own. Nonetheless, if
in a curriculum, a new training task $c'$ is generated via additive noise on a task $c$ in which the agent demonstrates good performance, the property is clearly exploited. Furthermore, policy representations $\pi(a|s,c,\omega)$ such as e.g. (deep) neural networks also tend to encode continuity w.r.t. $c$. We wanted to highlight these observations, as they allow to judge whether a given CRL algorithm, as well as ours, is applicable to given problem.

4. A Probabilistic Interpretation of Self-Paced Learning

We will now introduce a probabilistic view on self-paced learning, as discussed in Section 3.1. Our interpretation is based on viewing the variables $\nu_i$ as probabilities of a distribution over samples. More precisely, we define the categorical probability distribution $p(c=i|\nu) = \nu_i$ over the variable $c \in [1,N]$. For this to be a valid probability distribution, we need to introduce a 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 an expectation

$$\nu^*, \omega^* = \arg \min_{\nu, \omega} r(\omega) + \mathbb{E}_{p(c|\nu)} [f(x_c, y_c, \omega)] + \sum_{i=1}^{N} f(\alpha, p(c=i|\nu)), \quad \alpha > 0$$

s.t. $\sum_{i=1}^{N} \nu_i = 1$. (8)

The only difference w.r.t. (1) is the constraint that forces the variables $\nu_i$ to sum to 1. Unfortunately, depending on the self-paced regularizer $f(\alpha, p(c=i|\nu))$, the constraint yields unreasonable behavior. 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 $\nu^*$ to (8) simply puts all weight on the sample with the minimum loss, regardless of the value of $\alpha$.

Although there seems to be no general connection between objective (1) and (8) that holds for arbitrary self-paced regularizers, we will now investigate the regularizer

$$f_{KL,i}(\alpha, \nu_i) = \alpha \nu_i (\log(\nu_i) - \log(\mu(c=i))) - \alpha \nu_i,$$ (9)

where the probability distribution $\mu(c)$ is chosen by the user. We will shortly focus on this distribution in more detail. It is important to note that, due to the term $\mu(c=i)$, there is now an individual regularizer $f_{KL,i}$ for each sample. This formulation in line with the theory established by Meng et al. (2017) and simply corresponds to an individual regularizer $F_{\alpha,i}(\cdot)$ for each sample in (3). It is easily verified that $f_{KL,i}(\alpha, \nu_i)$ is convex w.r.t. $\nu_i$ and that the solution for objective (1)

$$\nu_{KL,i}(\alpha, l) = \mu(c=i) \exp \left( -\frac{1}{\alpha} \right)$$ (10)

fulfills the remaining axioms of Jiang et al. (2015) except for $\lim_{l \to 0} \nu_{KL,i}(\alpha, l) = 1$, since $\lim_{l \to 0} \nu_{KL,i}(\alpha, l) = \mu(c=i)$. Consequently, the distribution $\mu(c)$ represents a target distribution of samples under which the model should perform well. Indeed, we could simply remove the log-likelihood term $\log(\mu(c=i))$ from $f_{KL,i}(\alpha, \nu_i)$ and pre-weight each sample
with \( \mu(c=i) \), which would yield exactly the same curriculum while fulfilling all axioms. We stick to the form (9), as it soon allows us to connect \( f_{KL,i} \) to the KL divergence between \( p(c|\nu) \) and \( \mu(c) \). When employing \( f_{KL,i} \) and solving objective (8) w.r.t \( \nu \) (i.e. with the additional normalization constraint), the solution is given by

\[
\nu_{KL,i}^*(\alpha, l_i) = \frac{\mu(c=i) \exp\left(\frac{-1}{\alpha} l_i \right)}{\sum_{j=1}^{N} \mu(c=j) \exp\left(\frac{-1}{\alpha} l_j \right)}.
\]

Comparing (10) and (11), we see that the additional constraint \( \sum_{i=1}^{N} \nu_i = 1 \) simply rescales the variables \( \nu_{KL,i}^* \) by a constant. Since \( \mathbb{E}_{p(c|\nu)} [f(x_c, y_c, \omega)] = \sum_{i=1}^{N} \nu_i f(x_i, y_i, \omega) \), we see that consequently the only difference between (1) and (8) for this particular regularizer is a different weighting of the regularization term \( r(\omega) \) throughout the iterations of SPL. Further, if \( r(\omega) = 0 \), the two approaches are exactly equivalent, since a constant scaling does not change the location of the optima w.r.t \( \omega \) in both (1) and (8). Computing the functional form of the corresponding non-convex regularizer

\[
F_{KL,\alpha,i}(f(x_i, y_i, \omega)) = \int_0^{f(x_i, y_i, \omega)} \nu_{KL,i}^*(\alpha, l) \, dl = \mu(c=i) \alpha \left(1 - \exp\left(-\frac{1}{\alpha} \right)\right),
\]

we can conclude this section with a theorem that allows us to transfer the self-paced learning paradigm to the domain of RL.

**Theorem 1** Alternatingly solving

\[
\min_{\omega, \nu} \mathbb{E}_{p(c|\nu)} [f(x_c, y_c, \omega)] + \alpha D_{KL}(p(c|\nu) \parallel \mu(c))
\]

w.r.t. \( \omega \) and \( \nu \) is an MM scheme applied to the regularized objective

\[
\min_{\omega} \mathbb{E}_{\mu(c)} \left[ \alpha \left(1 - \exp\left(-\frac{1}{\alpha} f(x_c, y_c, \omega)\right)\right)\right].
\]

The proof of the theorem is straightforward given the observations made so far and simply requires to relate \( \sum_{i=1}^{N} f_{KL,i}(\alpha, \nu_t) \) to \( \alpha D_{KL}(p(c|\nu) \parallel \mu(c)) \), as done in the appendix. Figure 1 provides an intuition about the effect of the regularizer (12). As can be seen, \( F_{KL,\alpha,i}(l) \) exhibits a squashing effect to limit the attained loss \( l \) to a maximum value of \( \alpha \). The closer the non-regularized loss \( l \) attains this maximum value of \( \alpha \), the more it is treated as a constant value by \( F_{KL,\alpha,i}(l) \). Further away from \( \alpha \), a change in the non-regularized loss \( l \) leads to an increasingly linear change in the regularized loss \( F_{KL,\alpha,i}(l) \).

### 5. Application to Reinforcement Learning

For the remainder of this paper, we will turn to the problem of Reinforcement Learning (RL), where the self-paced learning scheme has so far found only a limited application in improving the regression performance while approximating a value function (Ren et al., 2018). The observations of the self-paced learning algorithm with the regularizer \( f_{KL,i} \) as well as Theorem 1 motivate another application of self-paced learning to generate a sequence...
of distributions over training tasks for an RL agent. We present an approximate scheme that follows this sequence of distributions and that we will later connect to the RL-as-inference paradigm. Subsequently, we perform experimental evaluations of the proposed curriculum learning scheme, showing its usefulness by matching and surpassing the performance of state-of-the-art curriculum learning approaches for RL.

5.1 Learning on an Infinite Set of Samples

At this point, adapting the self-paced learning approach to RL is as simple as realizing that the contextual RL objective (6) is directly connected to the expected loss over samples, i.e. that in the reinforcement learning setting $\mathbb{E}_{\mu(c)} [J(\omega, c)]$ takes the role of $\mathbb{E}_{\mu(c)} [f(x_c, y_c, \omega)]$. The only difference is that in RL it is both common to investigate distributions $\mu$ over continuous and discrete sets. For our algorithms and experiments, we investigate the continuous case. However, an application to the discrete case is possible as well. Having made this observation, we now propose a self-paced reinforcement learning objective

$$\min_{\omega} \mathbb{E}_{\mu(c)} [F_{KL,\alpha}(−J(\omega, c))] , \quad F_{KL,\alpha}(l) = \alpha \left(1 - \exp\left(-\frac{1}{\alpha}l\right)\right).$$

(13)

The negation of $J(\omega, c)$ is due to the RL objective being formulated as a maximization problem while the investigation of SPL was based on minimization problems. Without loss of generality, we furthermore assume that $J(\omega, c) \leq 0$ and hence $−J(\omega, c) \geq 0$. With the intuition gained from Figure 1, we know that this regularized objective down-weights the "importance" of tasks in which the agent currently achieves a low-reward. More importantly, however, Theorem 1 motivates the alternative objective

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

This, in turn, directly implies a curriculum for an RL algorithm as the sequence of distribution $p(c|\nu_i)$. This view as training the RL agent on the distribution $p(c|\nu_i)$ is especially
beneficial because it addresses a fundamental difference between reinforcement- and supervised learning. That is, to compute a gradient in a supervised learning task, no data needs to be gathered. To compute a gradient w.r.t. objective (13), we, however, are heavily relying on data collected by executing the current policy $\pi(\cdot|\cdot, c, \omega)$ in task $c$. Given the cost of collecting this data, the chosen scheme of data aggregation, i.e. the decision on which tasks contribute samples at which point during the learning, is of utter importance, and the connection between SPL and the distribution $p(c|\nu)$ directly tells us which tasks are relevant w.r.t. the gradient of the regularized RL objective (13) for a given value of $\alpha$. Intuitively, sampling $p(c|\nu)$ avoids tasks $c$ in which the current policy performs poorly and hence have a low influence on the gradient of (13) w.r.t. $\omega$.

### 5.2 Practical Considerations

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 $p(c|\nu_i)$ and $p(c|\nu_{i+1})$, yielding

$$\max_{\omega, \nu} \mathbb{E}_{p(c|\nu)} [J(\omega, c)] - \alpha D_{\text{KL}} (p(c|\nu) \parallel \mu(c))$$

subject to

$$D_{\text{KL}} (p(c|\nu) \parallel p(c|\nu')) \leq \epsilon$$

with $\nu'$ being the previously computed variables. In a practical algorithm, this secondary regularization is important because the expected performance $J(\omega, c)$ is approximated by a learned value function, which may not predict accurate values for contexts not likely under $p(c|\nu')$. The KL divergence constraint helps to avoid exploiting these false estimates too greedily. Furthermore, it enforces the observations made at the end of Section 3.2, as it forces the distribution over contextual variables, and hence tasks, to gradually change. From a theoretical perspective on SPL, the constraint clearly changes the form of $\nu^*$ making it not only dependent on $\alpha$ and $\omega$, but also on the previous parameter $\nu'$. 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 (14) to the RL-as-inference perspective in Section 8, 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.

### 6. Application to Episodic Reinforcement Learning

We first turn towards a particular perspective on the RL problem, which more closely resembles the supervised learning setting of Section 4, however, where there is no access to gradients of the objective function w.r.t. $\omega$.

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

$$r(\theta, c) = \mathbb{E}_{p_0,c(s)} [V_{\pi(a|s,\theta)}(s, c)]$$

(15)
we see that we can simply interpret \( r(\theta, c) \) as a function that, due to its complicated nature, does only allow for noisy observation of its function value without any gradient information. The noise in function observations arises from the fact that a rollout of policy \( \pi(a|s, \theta) \) in a context \( c \) corresponds to approximating the expectations in \( r(\theta, c) \) with a single sample. Approaches maximizing such functions are commonly referred to as black-box optimizers. Such a formulation of the RL problem has been shown to be effective for learning or adjusting behavioral policies in robotic scenarios (Kupcsik et al., 2013; Parisi et al., 2015), especially when carefully designing the policy \( \pi(a|s, \theta) \) to ensure safe behavior while using only a low-dimensional parameterization \( \theta \). One reason for the effectiveness is that the exploration of the algorithm is performed on the parameters \( \theta \) instead of on the actions \( a \). If the policy parameterization is well-chosen for the task, this form of exploration can be much more effective. The contextual relative entropy policy search (C-REPS) algorithm (Neumann, 2011; Kupcsik et al., 2013; Parisi et al., 2015) frames the maximization of (15) over a task distribution \( \mu(c) \) as a repeated entropy-regularized optimization

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

where \( p(\theta, c) = p(\theta|c)\mu(c) \) is the distribution obtained in the previous iteration. The particular form of the C-REPS algorithm allows for a straightforward incorporation of SPL, simply replacing the constraint \( \int q(\theta, c) d\theta = \mu(c) \) by a penalty term on the KL divergence between \( q(c) \) and \( \mu(c) \)

\[
\max_{q(\theta,c)} \mathbb{E}_{q(\theta,c)} \left[ r(\theta, c) \right] - \alpha D_{\text{KL}}(q(c) \| \mu(c)) \\
\text{s.t.} \quad D_{\text{KL}}(q(\theta, c) \| p(\theta, c)) \leq \epsilon. \tag{16}
\]

The above objective does not yet include the parameters \( \omega \) or \( \nu \) of the policy or the context distribution to be optimized, because both C-REPS, and also our implementation of SPL for episodic RL, solves the above optimization problem analytically to obtain a re-weighting scheme for samples \( (\theta_i, c_i) \sim p(\theta|c, \omega_k)p(c|\nu_k) \) based on the observed rewards \( r(\theta_i, c_i) \). The next parameters \( \omega_{k+1} \) and \( \nu_{k+1} \) 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 (16) analytically using the technique of Lagrangian multipliers, we obtain the following form for the variational distributions

\[
q(\theta, c) \propto p(\theta, c|\omega_k, \nu_k) \exp \left( \frac{r(\theta, c) - V(c)}{\eta_q} \right) = p(\theta, c|\omega_k, \nu_k) \exp \left( \frac{A(\theta, c)}{\eta_q} \right), \tag{17}
\]

\[
q(c) \propto p(c|\nu_k) \exp \left( \frac{V(c) + \alpha (\log(\mu(c)) - \log(p(c|\nu_k)))}{\eta_q} \right) = p(c|\nu_k) \exp \left( \frac{\beta(c)}{\eta_q} \right), \tag{18}
\]
two approximations are introduced: First, the expectations w.r.t. $V$ form for the value function $D$ approximated by re-weighting the samples via weights $w$. As previously mentioned, in practice the algorithm has only access to a set $\mathcal{D} = \{(\theta_i, c_i, r_i) | i \in [1, M]\}$.

The derivation of the dual objective as well as the solution to objective (16) are shown in the appendix. As previously mentioned, in practice the algorithm has only access to a set of samples $\mathcal{D} = \{(\theta_i, c_i, r_i) | i \in [1, M]\}$ and hence the analytic solutions (17) and (18) are approximated by re-weighting the samples via weights $w_i$. To compute the optimal weights $w_i$, the multipliers $V^\ast, \eta^\ast_q$ and $\eta^\ast_{\bar{q}}$ need to be obtained by minimizing the dual (19), to which two approximations are introduced: First, the expectations w.r.t. $p(\theta, c | \nu, \omega)$ are replaced by a sample-estimate from the collected samples in $\mathcal{D}$. Second, we introduce a parametric form for the value function $V(c) = \chi^T \phi(c)$ with a user-chosen feature function $\phi(c)$, such that we can optimize (19) w.r.t. $\chi$ instead of $V$.

After finding the minimizers $\chi^\ast, \eta^\ast_q$ and $\eta^\ast_{\bar{q}}$ of (19), the weights $w_i$ are then given by the exponential terms in (17) and (18). The new policy- and context distribution parameters are fitted via maximum likelihood to the set of weighted samples. To account for the error that originates from the sample-based approximation of the expectations in (19), we enforce the KL divergence constraint $D_{KL}(p(\theta, c | \nu_k, \omega_k) \parallel q(\theta, c | \omega_{k+1}, \nu_{k+1})) \leq \epsilon$ when updating the policy and context distribution. Again, details on this maximum likelihood step can be found in the appendix.

As was the case for the Supervised Learning setting, we need to choose a schedule for the value of $\alpha$. In a Supervised Learning scenario, it is preferable to choose as small step-sizes as possible in order to gradually transform the objective from an easy version towards to the target one. In Reinforcement Learning, each iteration of an RL algorithm

---

**Algorithm 1** Self-Paced Episodic Reinforcement Learning (SPRL)

**Input:** Initial context distribution- and policy parameters $\nu_0$ and $\omega_0$, Target context distribution $\mu(c)$, KL penalty proportion $\zeta$, Offset $K_\alpha$, Number of iterations $K$, Rollouts per policy update $M$, Relative entropy bound $\epsilon$

**for** $k = 1$ **to** $K$ **do**

**Collect Data:**
- Sample contexts: $c_i \sim p(c | \nu_{k-1}), i \in [1, M]$
- Sample and execute actions: $\theta_i \sim p(\theta | c_i, \omega_{k-1})$
- Observe reward: $r_i = r(\theta_i, c_i)$
- Create sample set: $\mathcal{D}_k = \{(\theta_i, c_i, r_i) | i \in [1, M]\}$

**Update Policy and Context Distributions:**
- Update schedule: $\alpha_k = 0$, if $k \leq K_\alpha$, else $B(\nu_{k-1}, \mathcal{D}_k)$
- Optimize dual function: $[\eta^*_{q}, \eta^*_{\bar{q}}, V^*] \leftarrow \text{arg min} \mathcal{G}(\eta_q, \eta_{\bar{q}}, V)$
- Calculate sample weights: $[w_i, \tilde{w}_i] \leftarrow [\exp\left(\frac{A(\theta_i, c_i)}{\eta_q}\right), \exp\left(\frac{\beta(c_i)}{\alpha + \eta_{\bar{q}}}\right)]$
- Infer new parameters: $[\omega_k, \nu_k] \leftarrow \{(w_i, \tilde{w}_i, \theta_i, c_i) | i \in [1, M]\}$

**end for**

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

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

(19)
Self-Paced Learning in RL 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 $\alpha$. In our implementation, the parameter $\alpha$ is chosen such that the KL divergence penalty w.r.t. the current context distribution is in constant proportion $\zeta$ to the average reward obtained during the last iteration of policy optimization

$$
\alpha_k = B(\nu_k, D_k) = \zeta \frac{1}{M} \sum_{i=1}^{M} r(\theta_i, c_i) \frac{D_{KL}(p(c|\nu_k) \parallel \mu(c))}{D_{KL}(p(c|\nu_k) \parallel \mu(c))}.
$$

We further set $\alpha$ to zero for the first $K_\alpha$ iterations. In combination with an initial context distribution $p(c|\nu_0)$ covering large parts of the context space, this choice of $\alpha$ 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. Algorithm 1 summarizes the overall procedure.

Before presenting experimental results, we want to briefly take a moment to investigate the behavior of the context distribution for the case $\alpha = 0$. For $\alpha = 0$, the update according to (18) reduces to $q(c) \propto p(c|\nu) \exp(V(c)/\eta_q)$. As e.g. highlighted in (Deisenroth et al., 2013), the term $V(c)$ can be interpreted as a contextual-value function, representing a soft-max operator over the expected reward under the policy $p(\theta|c, \omega)$

$$
V(c) = \eta \log \left( \int p(\theta|c, \omega) \exp \left( \frac{r(\theta, c)}{\eta} \right) d\theta \right).
$$

Repeating above update for a fixed $\omega$ until convergence would result in a distribution that puts all probability mass on the context with maximum expected value, which intuitively makes sense, since for $\alpha \to 0$, $\nu_{K_L,i}$ in (11) also collapses to a distribution that puts all probability mass on the sample with the smallest loss.

### 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 $\mu(c)$ 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...
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 $p(c|\nu)$ (colored areas) of one SPRL run together with the target distribution $\mu(c)$ (black line). Right: Task visualizations for different for different gate positions and widths. The crosses mark the corresponding positions in the context space.

function $V(c)$, while the policy $p(\theta|c, \omega) = \mathcal{N}(\theta|A_\omega \phi(c), \Sigma_\omega)$ uses linear features $\phi(c)$. 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 $p(c|\nu_0)$ that, in combination with setting $\alpha = 0$ for the first $K\alpha$ iterations, allows the algorithm to automatically choose the initial tasks on which learning should take place. After the first $K\alpha$ iterations, we then choose $\alpha$ 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 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 $c$. If the point-mass crashes into the wall, the experiment is stopped and the reward computed based on the current position. The reward function is an exponential of the 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 the gate) and 0, while all

¹. Code is publicly available under https://github.com/psclklnk/self-paced-rl
other parameters are controlled by the policy $\pi$, making $\theta$ a 14-dimensional vector.

We evaluate two setups in this gate environment, which differ in their target context distribution $\mu(c)$: 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 visualizes the obtained rewards for the investigated algorithms, the evolution of the sampling distribution $\tilde{\mu}(c)$ as well as tasks from the environment. In the “global” setting, we can see that SPRL converges significantly faster to the optimum 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. 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 the algorithm 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 in order to reach the goal without collisions. To simplify the visualization of the task distribution, we only allow two of the four obstacles to vary in size. The sizes of those two obstacles make up a task $c$ 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 $\pi$ encodes a ProMP (Paraschos et al., 2013), from which movements are sampled during training. In this task, $\theta$ 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 and more variance. 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. This behavior also explains the initially lower performance of SPRL on the target task compared to C-REPS and CMA-ES. 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 the 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 again represents a ProMP encoding the desired position of the first, third and fifth joint of the robot. Obviously, achieving the desired task with a poor initial policy is an unlikely event, leading to mostly uninformative rewards and hence a 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 progress by first training with larger cups and only progressively increasing the precision of the movement. Having access to only 16 samples per iteration,
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 $\tilde{\mu}(c)$ at different iterations (colored areas) of one SPRL run together with the target distribution (black line). Right: Task visualization on the real robot (upper) and in simulation with a scale of 2.5 (lower).

the algorithms did not always learn to achieve the task. However, the final policies learned by SPRL clearly 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 previous experiments demonstrate that the self-paced learning paradigm can indeed be beneficial in the episodic RL setting, so that as a next step we want to investigate its application when using a stochastic policy of the form $\pi(a|s, c, \omega)$. In the previous section we compared to the GoalGAN and SAGG-RIAC algorithm. A desirable property of these algorithms is that they can be combined with arbitrary RL algorithms to generate a curriculum for those. The implementation of our episodic scheme, however, was tied to the REPS-like optimization procedure. In the step-based RL setting, this can be disadvantageous, since such algorithms usually need to make more and possibly coarser approximations, such that depending on the task, one or the other RL algorithm may be more appropriate for a task. Due to the possibility of updating the SPL objective in a block-coordinate manner w.r.t. $\omega$ and $\nu$, it is straightforward to use arbitrary algorithms that perform the optimization w.r.t. $\omega$ under a given $\nu$ and then use the SPL scheme to update the context distribution given $\omega$. In the following section, we will detail on an approximate implementation which allows to create curricula following the SPL paradigm for arbitrary RL algorithms by making use of the value functions that they estimate during policy optimization.

7.1 Algorithmic Implementation

When optimizing (14) w.r.t the policy parameters $\omega$ using an RL algorithm of choice, a dataset of trajectories is generated

$$D_k = \{(c_i, \tau_i)| c_i \sim p(c|\nu_k), \tau_i \sim p(\tau|c_i, \omega_k), i \in [1, M]\}.$$

In the above definition, a trajectory is defined as

$$\tau_i = \{(s_{i,j}, a_{i,j}, r_{i,j})| a_{i,j} \sim p(a|s_{i,j}, c_i, \omega), s_{i,j+1} \sim p(s_{i,j+1}|s_{i,j}, a_{i,j}), s_{i,0} \sim p_0, c_i(s), j=1, \ldots\}.$$
Algorithm 2: Self-Paced Deep Reinforcement Learning (SPDL)

**Input:** Initial context distribution- and policy parameters $\nu_0$ and $\omega_0$, Target context distribution $\mu(c)$, KL penalty proportion $\zeta$ and offset $K_\alpha$ OR expected performance threshold $V_{LB}$, Number of iterations $K$, Rollouts per policy update $M$, Relative entropy bound $\epsilon$

**for** $k = 1$ **to** $K$ **do**

**Agent Improvement:**
Sample contexts: $c_i \sim p(c|\nu_k), \ i \in [1,M]$
Rollout trajectories: $\tau_i \sim p(\tau|c_i,\omega_k), \ i \in [1,M]$
Obtain $\omega_{i+1}$ from RL algorithm of choice using $D_k = \{(c_i,\tau_i)|i \in [1,M]\}$
Use or Estimate $V_{\omega_{k+1}}(s_i,0,c_i)$ for contexts $c_i$

**Context Distribution Update:**
**IF** $k \geq K_\alpha$ **OR** $\frac{1}{M} \sum_{i=1}^{M} V_{\omega_{k+1}}(s_i,0,c_i) < V_{LB}$: Obtain $\nu_{k+1}$ from (21) with $\alpha_k = 0$
**ELSE-IF** $k > K_\alpha$: Obtain $\nu_{k+1}$ optimizing (21), using $\alpha_k = B(\nu_k,D_k)$ (20)
**ELSE:** Obtain $\nu_{k+1}$ optimizing (22)

**end for**

One unifying property of many RL algorithms, is their reliance on estimating the state-value function $V_\omega(s_0,c)$, each in their respective way, as a proxy to optimizing the policy. We show how it is possible to use this estimate in the curriculum learning framework, similar to what we have discussed in the former sections. Notice that the main objective in Equation (6) can be written as $J(\omega,c) = \mathbb{E}_{p_0,c(s_0)}[V_\omega(s_0,c)]$. For a given context $c_i$, we can coarsely approximate the expectation w.r.t. $p_0,c(s_0)$ with the initial state $s_{i,j}$ contained in the set of trajectories. This yields an approximate form of (14) given by

$$
\max_{\nu_{k+1}} \frac{1}{M} \sum_{i=1}^{M} \frac{p(c_i|\nu_{k+1})}{p(c_i|\nu_k)} V_\omega(s_{i,0},c_i) - \alpha_k D_{KL}(p(c|\nu_{k+1}) \parallel \mu(c))
$$

s.t. $D_{KL}(p(c|\nu_{k+1}) \parallel p(c|\nu_k)) \leq \epsilon$. (21)

The first term in Objective (21) is an approximation to $\mathbb{E}_{p(c|\nu_{k+1})}[J(\omega,c)]$ 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).

**Alternative $\alpha$-schedule:** The evaluation in the previous setting showed that choosing $\alpha_k$ in each iteration according to (20) was sufficient to improve the performance of the learner in the investigated experiments. However, the introduced schedule requires an offset parameter $K_\alpha$ as well as a penalty proportion $\zeta$ to be specified. In the following experiments, we additionally explore another approach of choosing $\alpha_k$. This approach only requires to specify an expected level of performance $V_{LB}$ that the agent should not fall short of during training. Due to the decoupled optimization of the policy $\pi$ and the context

---

**Algorithm 2:** Self-Paced Deep Reinforcement Learning (SPDL)

**Input:**
- Initial context distribution and policy parameters $\nu_0$ and $\omega_0$.
- Target context distribution $\mu(c)$.
- KL penalty proportion $\zeta$ and offset $K_\alpha$ OR expected performance threshold $V_{LB}$.
- Number of iterations $K$.
- Rollouts per policy update $M$.
- Relative entropy bound $\epsilon$.

**for** $k = 1$ **to** $K$ **do**

**Agent Improvement:**
- Sample contexts: $c_i \sim p(c|\nu_k), \ i \in [1,M]$.
- Rollout trajectories: $\tau_i \sim p(\tau|c_i,\omega_k), \ i \in [1,M]$.
- Obtain $\omega_{i+1}$ from RL algorithm of choice using $D_k = \{(c_i,\tau_i)|i \in [1,M]\}$.
- Use or Estimate $V_{\omega_{k+1}}(s_i,0,c_i)$ for contexts $c_i$.

**Context Distribution Update:**
**IF** $k \geq K_\alpha$ **OR** $\frac{1}{M} \sum_{i=1}^{M} V_{\omega_{k+1}}(s_i,0,c_i) < V_{LB}$: Obtain $\nu_{k+1}$ from (21) with $\alpha_k = 0$.
**ELSE-IF** $k > K_\alpha$: Obtain $\nu_{k+1}$ optimizing (21), using $\alpha_k = B(\nu_k,D_k)$ (20).
**ELSE:** Obtain $\nu_{k+1}$ optimizing (22).

**end for**
distribution \( p(c|\nu) \), this can be easily realized by rewriting (14) as

\[
\min_{\nu} D_{KL} \left( p(c|\nu) \parallel \mu(c) \right)
\]

subject to

\[
\mathbb{E}_{p(c|\nu)} [J(\omega, c)] \geq V_{LB}
\]

\[
D_{KL} \left( p(c|\nu) \parallel p(c|\nu') \right) \leq \epsilon.
\]

The main modification is to avoid the explicit trade-off between expected agent performance and KL divergence by minimizing the KL divergence subject to a constraint on the expected agent performance. Investigating the Lagrangian of the above optimization problem

\[
L(\nu, \alpha, \eta) = D_{KL} \left( p(c|\nu) \parallel \mu(c) \right) + \alpha \left( V_{LB} - \mathbb{E}_{p(c|\nu)} [J(\omega, c)] \right)
+ \eta \left( D_{KL} \left( p(c|\nu) \parallel p(c|\nu') \right) - \epsilon \right),
\]

we see that the constraint reintroduces a scalar \( \alpha \) that trades-off the expected agent performance and KL divergence to \( \mu(c) \). The value of this scalar is, however, now automatically chosen to fulfill the imposed constraint on the expected agent performance. For an implementation of (22), we again replace \( J(\omega, c) \) by an importance-sampled monte carlo estimate as done in (21).

At this point, we have replaced the parameter \( \zeta \) by \( V_{LB} \). The benefit is a more intuitive choice of this hyperparameter, since it is directly related to the quantity being optimized. Furthermore, we can easily remove the need for the offset parameter \( K_\alpha \) by setting \( \alpha_k = 0 \) in (21) until we first reach \( V_{LB} \). From then on, we compute the new context distribution by optimizing (22). 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.

Algorithm 2 outlines the two resulting variations of SPL for step-based RL algorithms. We now compare this algorithm to different state-of-the-art curriculum learning algorithms for RL.

### 7.2 Experiments

Besides an improved learning performance, one motivation for Algorithm 2 is the independence of the underlying RL algorithm. To investigate these two properties, we evaluate SPDL in three different environments with different DRL algorithms. We evaluate the performance using TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017) and SAC (Haarnoja et al., 2018). For all DRL algorithms, we use the implementations provided in the Stable Baselines library (Hill et al., 2018).  

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, i.e. we only investigate the “precision” setting of the point-mass environment. The final environment is a robotic ball-catching environment.

2. Code for running the experiments can be found at https://github.com/psclklnk/spdl
This environment constitutes a shift in curriculum paradigm as well as reward function. Instead of guiding learning towards a specific target task, this third environment requires to learn a ball-catching policy over a wide range of initial states (ball position and velocity). The reward function is sparse compared to the dense ones employed in the first two environments. To judge the performance of SPDL, we compare the obtained results to state-of-the-art CRL algorithms ALP-GMM (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 \( c \in \mathbb{R}^3 \) 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 \( \mu(c) \) is a narrow Gaussian with 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 context space and one in which the friction parameter is fixed to its target value of 0. As Figure 8 and Table 1 indicate, SPDL significantly increases the asymptotic reward on the target task compared to other methods. We see that the episodic version (SPRL), which we applied by defining the episodic RL policy \( p(\theta|c, \omega) \) to choose the weights \( \theta \) of the policy network for a given context \( c \), learns much more slowly compared to its step-based counterpart. Increasing the dimension of the context space has a negative impact on the performance of the other CL algorithms. For SPDL, there is no statistically significant difference in performance across the two settings. We suspect that this effect arises because both ALP-GMM and GoalGAN have no notion of a target distribution. Consequently, for a context distribution \( \mu(c) \) with negligible variance, a higher context dimension decreases the average proximity of sampled tasks to the target one. By having a notion of a target distribution, SPDL ultimately samples contexts that
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 $p(c)$ 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 $\mu(c)$.

are close to the desired ones, regardless of the dimension. The context distributions $p(c)$ 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 pace 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 somewhat consistent with the one observed in Section 6.2.1.

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). The goal is to reach the other side of a wall by passing through a gate, whose width and position is determined by the contextual variable $c \in \mathbb{R}^2$ (Figure 7).

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

3. We use the Nvidia Isaac Gym simulator (Nvidia, 2019) for this experiment.
behavior is unreliable and inefficient, causing the action penalties in combination with the discount factor to prevent this better behavior from being reflected in the reward.

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 it for excessive movements. In the simulated environment, the ball is considered caught if it is in contact with the end effector. The context $c \in \mathbb{R}^3$ 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. In this environment, the context $c$ is not visible to the policy, as it only changes the initial state distribution $p(s_0)$ 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, which can be leveraged by CL algorithms.

Since SPDL and GoalGAN support to specify the initial context distribution, we investigate
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.

|                | 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           | 8.45 ± 0.4| 6.85 ± 0.8| 8.94 ± 0.1| 5.67 ± 0.8| 47.0 ± 2.0| 53.9 ± 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|

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 does not significantly change the performance in this task. The context distributions $p(c)$ 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 with differing pace, as can be seen in the visualizations of $p(c)$ in Figure 9 for iteration 200: Two of the three distributions fully match the target distribution while the third only covers half of it.

### 7.2.4 Schedule Comparison

So far we have not evaluated the performance of the schedule for $\alpha$ that is based on a lower bound $V_{LB}$ on the expected agent performance during training. The previously reported results were achieved with the heuristic based on a chosen offset $K_\alpha$ and a chosen penalty percentage $\zeta$. The final rewards achieved with the two heuristics are compared in Table 2. The results were not significantly different according to Welch’s t-test with $p < 1\%$, indicating that the much simpler heuristic performs just as well in the investigated environments.
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.

|                | SPDL($K_\alpha, \zeta$) | SPDL($V_{LB}$) | SPDL($K_\alpha, \zeta$)* | SPDL($V_{LB}$)* |
|----------------|---------------------------|----------------|---------------------------|-----------------|
| TRPO (P3D)     | 8.04 ± 0.25               | 7.79 ± 0.28    | -                         | -               |
| PPO (P3D)      | 8.45 ± 0.42               | 8.66 ± 0.07    | -                         | -               |
| SAC (P3D)      | 6.85 ± 0.77               | 7.13 ± 0.71    | -                         | -               |
| TRPO (P2D)     | 7.47 ± 0.33               | 7.67 ± 0.2     | -                         | -               |
| PPO (P2D)      | 8.94 ± 0.10               | 9.01 ± 0.07    | -                         | -               |
| SAC (P2D)      | 5.67 ± 0.77               | 6.56 ± 0.82    | -                         | -               |
| PPO (ANT)      | 1371 ± 23                 | 1305 ± 38      | -                         | -               |
| TRPO (BC)      | 47.0 ± 2.0                | 50.0 ± 1.5     | 43.3 ± 2.0                | 46.0 ± 1.5      |
| PPO (BC)       | 53.9 ± 0.4                | 51.6 ± 1.7     | 49.3 ± 1.4                | 51.8 ± 0.5      |
| SAC (BC)       | 34.1 ± 2.3                | 34.1 ± 1.3     | 36.9 ± 1.0                | 37.1 ± 1.2      |

8. An Inference Perspective on Self-Paced Reinforcement Learning

As noted in Section 5, the KL divergence regularization w.r.t. $\nu$ in (14) was done with the goal of stabilizing 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 MM 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).

8.1 RL as Inference

To establish 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 $O \in \{0, 1\}$ that depends on the sum of rewards along a trajectory of states and actions $\tau = \{(s_t, a_t)|t = 0, 1, \ldots\}$

$$p(O|\tau, c) \propto \exp \left( r(\tau, c) \right) = \exp \left( \sum_{t=0}^{\infty} r_c(s_t, a_t) \right).$$

(23)

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

$$p(\tau|c, \omega) = p_0,c(s_0) \prod_{t \geq 0} \tilde{p}_c(s_{t+1}|s_t, a_t)\pi(s_t|a_t, c, \omega)$$

(24)

this admits the definition of the likelihood of event $O$ as the expectation over all trajectories generated by the policy with parameters $\omega$ in a given MDP $\mathcal{M}(c)$

$$p(Y|X, z, \omega) = p(O|c, \omega) = \int p(O|\tau, c)p(\tau|c, \omega)\ d\tau.$$
The transition probabilities $\tilde{p}_c$ in (24) are a modified version of the original transition probabilities $p_c$ that introduce a “termination” probability in each step that can occur with a probability of $1 - \gamma$ (see (Levine, 2018) for details), which introduces the concept of a discounting factor $\gamma$ into the probabilistic model. Introducing a context distribution $p(c|\nu)$ then yields a probabilistic interpretation of the contextual RL objective into the probabilistic model. Introducing a context distribution $p(\nu)$ then yields a probabilistic interpretation of the contextual RL objective

$$p(O|\omega, \nu) = \int p(O|\tau, c)p(\tau|c, \omega)p(c|\nu) \, dc \, d\tau. \quad (26)$$

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

\begin{align*}
\text{E-Step : } q^k(\tau, c) &= \arg\min_{q(\tau, c)} D_{KL}\left(q(\tau, c) \mid\mid p(\tau, c|O, \omega^k, \nu^k)\right) \quad (27) \\
\text{M-Step : } \omega^{k+1}, \nu^{k+1} &= \arg\max_{\omega, \nu} \mathbb{E}_{q^k(\tau, c)}[\log(p(O, \tau, c|\omega, \nu))] . \quad (28)
\end{align*}

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

### 8.2 Connection to Self-Paced Reinforcement Learning

At this point, two simple reformulations are required to establish the connection between the KL-regularized objective (14) and the Expectation-Maximization algorithm on LVM (26). First, we can reformulate the the M-Step as an M-projection (i.e. a maximum-likelihood fit of the parametric model $q(\tau, c|\omega, \nu)$ to $q^k(\tau, c)$)

$$\arg\max_{\omega, \nu} \mathbb{E}_{q^k(\tau, c)}[\log(p(O, \tau, c|\omega, \nu))] = \arg\min_{\omega, \nu} D_{KL}\left(q^k(\tau, c) \mid\mid q(\tau, c|\omega, \nu)\right) ,$$

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

$$\arg\min_{q(\tau, c)} D_{KL}\left(q(\tau, c) \mid\mid p(\tau, c|O, \omega^k, \nu^k)\right)$$

$$= \arg\max_{q(\tau, c)} \mathbb{E}_{q(\tau, c)}[R(\tau, c)] - D_{KL}(q(\tau, c) \mid\mid p(\tau, c|\omega, \nu)) ,$$

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

$$\arg\max_{q(c)} \mathbb{E}_{q(c)}\left[\mathbb{E}_{p(\tau|c, \omega^k)}[R(\tau, c)]\right] - \alpha D_{KL}(q(c) \mid\mid \mu(c)) - D_{KL}(q(c) \mid\mid p(c|\nu^k)) . \quad (29)$$
This result clearly resembles (14), where however the optimization is carried out w.r.t. \( q(c) \) instead of \( \nu \) and the KL divergence w.r.t. \( p(c|\nu^k) \) is treated as a penalty term instead of a constraint. Not fitting the parameters \( \nu^{k+1} \) directly but in a separate (M-)step is also done by C-REPS and our episodic implementation. 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).

8.3 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 \( \alpha D_{KL} (q(c) \parallel \mu(c)) \) in the modified E-Step (29) results in an E-Step to a modified target distribution. That is

\[
\begin{align*}
\arg\min_{q(c)} D_{KL} \left( q(c) \parallel \frac{1}{Z} p(c|O, \omega^k, \nu^k) \right) + \alpha D_{KL} (q(c) \parallel \mu(c)) \\
= \arg\min_{q(c)} D_{KL} \left( q(c) \parallel \frac{1}{Z} p(c|O, \omega^k, \nu^k) \nu^{1+\alpha} \mu(c)^{1+\alpha} \right).
\end{align*}
\]

The modified target distribution in (30) is performing an interpolation between \( \mu(c) \) and \( p(c|O, \omega^k, \nu^k) \) based on the parameter \( \alpha \). Looking back at the sampling distribution induced by \( \nu_{KL,i}^\ast \) (Eq. (10))

\[
p(c|\alpha, \omega) \propto \mu(c) \exp \left( -J(\omega, c) \right)^\frac{1}{\alpha},
\]

we can see that, similarly to the modified E-Step, the distribution encoded by \( \nu_{KL,i}^\ast \) interpolates between \( \mu(c) \) and the distribution \( p(c) \propto \exp \left( -J(\omega, c) \right) \). 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 \( \mu(c) \) with many isolated modes of density (Kirkpatrick et al., 1983; Aluffi-Pentini et al., 1985; Marinari and Parisi, 1992; Ueda and Nakano, 1995). The main idea is to not directly apply inference methods to \( \mu(c) \) but to make use of a tempered distribution \( p_\alpha(c) \) which interpolates between \( \mu(c) \) and a user-chosen reference distribution \( \rho(c) \) from which samples can be easily drawn by the employed inference method (e.g. a Gaussian distribution). Doing repeated inference for varying values of \( \alpha \) allows to explore the isolated modes more efficiently and with that yielding more accurate samples from \( \mu(c) \). Intuitively, initially sampling from \( \rho(c) \), chosen to be free from isolated modes, and gradually progressing towards \( \mu(c) \) while using the previous inferences as initializations avoids getting stuck in poor local optima of \( \mu(c) \). This technique makes the inference algorithm less dependent on a good initialization.

We can easily identify both (30) and (31) to be particular tempered distributions \( p_\alpha(c) \). There, however, seems to be a striking difference to aforementioned tempering scheme: The target density \( \mu(c) \) is typically trivial, not requiring any advanced inference machinery. However, although \( \mu(c) \) may be trivial from an inference perspective, the density \( p(\omega|O) \propto p(\omega) \int p(O|c, \omega) \mu(c) dc \), i.e. the posterior over policy parameters, is highly challenging for contexts \( c \) distributed according to \( \mu(c) \), as it can contain many modes that are
highly isolated, which is why tempering helps in achieving better performance when employed in combination with RL. For low values of $\alpha$, it is easier to find high-density modes of $p_\alpha(\omega|O) \propto p(\omega) \int p(O|c, \omega)p_\alpha(c) dc$. These modes can then be tracked while increasing the value of $\alpha$.

The connection of SPL and the concept of tempering yields interesting insights into the problem of choosing both a good schedule for $\alpha$ and also the general design of $p_\alpha(c)$. As introduced in Section 3.1, the particular choice of the self-paced regularizer $f(\alpha, \nu)$, and hence the regularizer $F_\alpha(l)$, is closely related to the particular form of $p_\alpha(c)$. 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 $p_\alpha$ has a tremendous effect on the error of Monte-Carlo estimates of ratios between normalization constants. Furthermore, they compute the optimal form of $p_\alpha$ 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 was the schedule of $\alpha$. The value of $\alpha$ should be increased as fast as possible while ensuring stability of the RL agent. We made use of a simple heuristic that accomplished that task sufficiently well. However, there may be a tremendous margin for improvement. In the inference literature, people readily investigated the problem of choosing $\alpha$, as they face a similar trade-off problem between required computation time of inference methods and ensuring usefulness of their results (Mandt et al., 2016; Graham and Storkey, 2017; Luo et al., 2018). Again, it may be possible to draw inspirations from these works to design better schedules for $\alpha$ in RL problems.

9. 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 $\mu(c)$ 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 above 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 $\alpha$ 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 8 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 (30) 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

As stated in the main text, we only need to relate the sum over the regularizers

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

to the KL divergence between $p(c|\nu)$ and $\mu(c)$. Remembering $p(c=i|\nu) = \nu_i$, it follows that

$$\sum_{i=1}^{N} f_{\text{KL},i}(\alpha, \nu_i) = \alpha \sum_{i=1}^{N} p(c=i|\nu) (\log(p(c=i|\nu)) - \log(\mu(c=i))) - \alpha \sum_{i=1}^{N} p(c=i|\nu)$$

$$= \alpha D_{\text{KL}}(p(c=i|\nu) \parallel \mu(c=i)) - \alpha,$$

since $\sum_{i=1}^{N} p(c=i|\nu) = 1$ per definition of a probability distribution. We hence see that we can simply remove the term $-\alpha \nu_i$ from the self-paced regularizer $f_{\text{KL},i}$ when enforcing $\sum_{i=1}^{N} p(c=i|\nu) = 1$, leaving us with the KL divergence.

Appendix B. Self-Paced Episodic Reinforcement Learning Derivations

This appendix serves to highlight some important details regarding the derivation of the weights (17) and (18) as well as the dual objective (19). The most notable detail is the introduction of an additional distribution $q(c)$ that takes the role of the marginal $\int q(\theta, c) \, d\theta$ as well as the regularization of this additional distribution via a KL divergence constraint.
w.r.t. to the previous marginal \( p(c) = \int p(\theta, c) \, d\theta \). This yields the following objective
\[
\max_{q(\theta,c),q(c)} \mathbb{E}_{q(\theta,c)} \left[ r(\theta,c) \right] - \alpha D_{KL}(q(c) \| \mu(c))
\]
subject to \( D_{KL}(q(c) \| p(c)) \leq \epsilon \)
\[
\int q(\theta, c) \, d\theta = 1
\]
\[
\int q(c) \, dc = 1
\]
\[
\int q(\theta, c) \, d\theta = q(c) \quad \forall c \in \mathcal{C}.
\]
However, these changes are purely of technical nature as they allow to derive numerically stable weights and duals. It is straightforward to verify that \( D_{KL}(q(\theta, c) \| p(\theta, c)) \leq \epsilon \)
implies \( D_{KL}(q(c) \| p(c)) \leq \epsilon \). Hence, the constraint \( \int q(\theta, c) \, d\theta = q(c) \) guarantees that a solution \( q(\theta, c) \) to above optimization problem is also a solution to (16). The dual as well as the updated weights now follow from the Lagrangian
\[
\mathcal{L}(q, V, \eta_q, \eta_{\tilde{q}}, \lambda_q, \lambda_{\tilde{q}}) = \mathbb{E}_{q(\theta,c)} \left[ r(\theta,c) \right] - \alpha D_{KL}(q(c) \| \mu(c))
\]
\[
+ \eta_q (\epsilon - D_{KL}(q(\theta, c) \| p(\theta, c))) + \lambda_q \left( 1 - \int q(\theta, c) \, d\theta \right)
\]
\[
+ \eta_{\tilde{q}} (\epsilon - D_{KL}(q(c) \| p(c))) + \lambda_{\tilde{q}} \left( 1 - \int q(c) \, dc \right)
\]
\[
+ \int V(c) \left( \int q(\theta, c) \, d\theta - q(c) \right) \, dc.
\]
Note that we slightly abuse notation and overload the argument \( q \) in the definition of the Lagrangian. The update equations (17) and (18) follow from the two conditions \( \frac{\partial \mathcal{L}}{\partial q(\theta,c)} = 0 \) and \( \frac{\partial \mathcal{L}}{\partial q(c)} \mathcal{L} = 0 \). Inserting (17) and (18) into equation (32) then allows to derive the dual (19). 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 (17) and (18) 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 dataset of \( N \) weighted samples
\[
D = \{(w_i^x, w_i^y, x_i, y_i) | i = 1, \ldots, N\},
\]
with \( x_i \in \mathbb{R}^{d_x}, y_i \in \mathbb{R}^{d_y} \), the task of fitting a joint-distribution
\[
q(x, y) = q_y(y|x)q_x(x) = \mathcal{N}(y|A\phi(x), \Sigma_y)\mathcal{N}(x|\mu_x, \Sigma_x)
\]
to \( D \) while limiting the change with regards to a reference distribution
\[
p(x, y) = p_y(y|x)p_x(x) = \mathcal{N}(y|\hat{A}\phi(x), \hat{\Sigma}_y)\mathcal{N}(x|\hat{\mu}_x, \hat{\Sigma}_x),
\]
self-paced learning in RL.
with feature function $\phi : \mathbb{R}^{d_x} \mapsto \mathbb{R}^o$, can be expressed as a constrained optimization problem

$$\max_{\mathbf{A}, \Sigma_y, \mu_x, \Sigma_x} \sum_{i=1}^{N} (w_i x_i \log(q_x(x_i)) + w_i y_i \log(q_y(y_i|x_i)))$$

s.t. $D_{KL}(p \parallel q) \approx \frac{1}{N} \sum_{i=1}^{N} D_{KL}(p_y(\cdot|x_i) \parallel q_y(\cdot|x_i)) + D_{KL}(p_x \parallel q_x) \leq \epsilon$

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 $\epsilon$ this is a reasonable approximation. Since the distributions $p_x, p_y, q_x$ and $q_y$ 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 $\eta$ and the samples from $D$

$$\mathbf{A} = \left[ \sum_{i=1}^{N} \left( w_i y_i + \frac{\eta}{N} \hat{\mathbf{A}} \phi(x_i) \right) \phi(x_i)^T \right]^{-1} \left[ \sum_{i=1}^{N} \left( w_i + \frac{\eta}{N} \right) \phi(x_i) \phi(x_i)^T \right],$$

$$\Sigma_y = \frac{\sum_{i=1}^{N} w_i \Delta y_i \Delta y_i^T + \eta \hat{\Sigma}_y + \frac{\eta}{N} \Delta \mathbf{A} \sum_{i=1}^{N} \phi(x_i) \phi(x_i)^T \Delta \mathbf{A}^T}{\sum_{i=1}^{N} w_i + \eta},$$

$$\mu_x = \frac{\sum_{i=1}^{N} w_i x_i + \eta \tilde{\mu}_x}{\sum_{i=1}^{N} w_i + \eta},$$

$$\Sigma_x = \frac{\sum_{i=1}^{N} w_i (x_i - \mu_x) (x_i - \mu_x)^T + \eta \left( \hat{\Sigma}_x + (\mu_x - \tilde{\mu}_x)(\mu_x - \tilde{\mu}_x)^T \right)}{\sum_{i=1}^{N} w_i + \eta},$$

with $\Delta y_i = y_i - \mathbf{A} \phi(x_i)$ and $\Delta \mathbf{A} = \mathbf{A} - \hat{\mathbf{A}}$. Above equations yield a simple way of enforcing the KL bound on the joint distribution: Since $\eta$ is zero if the constraint on the allowed KL divergence is not active, $\mathbf{A}, \Sigma_y, \mathbf{\mu}_x$ and $\Sigma_x$ can be first computed with $\eta = 0$ and only if the allowed KL divergence is exceeded, $\eta$ needs to be found by searching the root of

$$f(\eta) = \epsilon - \frac{1}{N} \sum_{i=1}^{N} D_{KL}(p_y(\cdot|x_i) \parallel q_y(\cdot|x_i)) + D_{KL}(p_x \parallel q_x),$$

where $q_y$ and $q_x$ are expressed as given by above formulas and hence implicitly depend on $\eta$. 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 to stay above a certain lower bound $\sigma_{LB}$ 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 $p(c|\nu)$, we simply clip the standard deviation until the KL divergence w.r.t. the target distribution $\mu(c)$ falls below a certain threshold $D_{KL,LB}$. This threshold was chosen such that the distribution with the clipped standard deviation already contains the target distribution. The specific values of $D_{KL,LB}$ and $\sigma_{LB}$ 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 ($s_f$ and $s_g$) is less than a given threshold $\tau$

$$\text{Success}(\theta, c) = \begin{cases} 1, & \text{if } \|s_f(\theta) - s_g(c)\|_2 < \tau, \\ 0, & \text{else.} \end{cases}$$

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 $\mathcal{N}(\theta|A\phi(c), \Sigma_\theta)$, where $\phi(c)$ 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 $\theta$ encode a feed-forward

|                      | $\epsilon$ | $n_{\text{samples}}$ | Buffer Size | $\zeta$ | $K_\alpha$ | $\sigma_{\text{LB}}$ | $D_{\text{KL,LB}}$ |
|----------------------|------------|-----------------------|-------------|----------|-------------|----------------------|---------------------|
| 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.
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.

|                      | $\delta_{\text{NOISE}}$ | $n_{\text{ROLLOUT GG}}$ | $n_{\text{GOALS}}$ | $n_{\text{HIST}}$ |
|----------------------|--------------------------|--------------------------|---------------------|-------------------|
| 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               |

policy by weighting several Gaussian basis functions over time

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

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.

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 $N_{\text{SAMPLES}} \times \text{BUFFER SIZE}$ samples, from which only $N_{\text{SAMPLES}}$ 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 $\delta_{\text{NOISE}}$ and the number of iterations that pass between the training of the GAN $n_{\text{ROLLOUT GG}}$ 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 $n_{\text{GOALS}}$ as well as the maximum number of recent samples for the competence computation $n_{\text{HIST}}$.

Tables 3 and 4 show the aforementioned hyperparameters of C-REPS, SPRL, GoalGAN and SAGG-RIAC for the different environments.

**Point-Mass Experiment**

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

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 5 \\ -1 \end{bmatrix} + u + \delta, \quad \delta \sim \mathcal{N}(0, 2.5 \times 10^{-3} I).$$
The columns show visualizations of the point-mass trajectories (upper plots) as well as the obtained rewards (lower plots) in the gate 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).

The point-mass is controlled by two linear controllers

$$C_i(x, y) = K_i \left[ \frac{x_i - x}{y_i - y} \right] + k_i, \quad i \in [1, 2], \quad K_i \in \mathbb{R}^{2 \times 2}, \quad k_i \in \mathbb{R}^2, \quad x_i, y_i \in \mathbb{R},$$

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 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 10 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 a reward close to the desired contexts, SPRL avoids this sub-optimal solution.
Reacher Experiment

In the Reacher experiment, the ProMP encoded by the policy \( \pi \) 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, \( \theta \) 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 \( \theta \) 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

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

where \( s = (x \ xdot \ y \ ydot) \) is the position and velocity of the end-effector, \( a = (a_x \ a_y) \) 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.

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, \( \theta \) is a 15-dimensional vector. The reward function is defined as

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

| Environment       | $K_\alpha$ | $\zeta$ | $K_{\text{OFFSET}}$ | $V_{\text{LB}}$ | $n_{\text{STEP}}$ | $\sigma_{\text{LB}}$ | $D_{\text{KL,LB}}$ |
|-------------------|------------|---------|----------------------|-----------------|-------------------|----------------------|-----------------|
| 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              | -                    | -              |

This encodes a preference over movements that deviate as less as possible from the initial position while still solving the task.

Looking back at Table 3, the value of $\zeta$ stands out, as it is significantly higher than in the other experiments. We suppose that such a large value of $\zeta$ 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 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 satisfactorily. 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 $n_{\text{STEP}}$ in the environment between context distribution updates instead of the number of trajectory rollouts. The additional parameter $K_{\text{OFFSET}}$ 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 $\omega_0$ in the algorithm sketched in the main paper. We selected the best $\zeta$ 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 hyperparam-
eters 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 $\alpha$-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

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

For GoalGAN we tuned the amount of random noise that is added on top of each sample $\delta_{NOISE}$, 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

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

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 $\mu$ and the vector of standard deviations for each dimension $\delta$. 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 $s = [x \dot{x} y \dot{y}]$. The actions correspond to the force applied in x- and y-dimension $a = [F_x F_y]$. 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 $c = [p_g \ w_g \ \mu_k] \in [-4, 4] \times [0.5, 8] \times [0, 4] \subset \mathbb{R}^3$. 

36
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.

| Environment          | \( p_{\text{RAND}} \) | \( n_{\text{ROLLOUT}} \) | \( s_{\text{BUFFER}} \) | \( \delta_{\text{NOISE}} \) | \( n_{\text{ROLLOUT}} \) | \( p_{\text{SUCCESS}} \) |
|----------------------|-------------------------|---------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 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                      |

The dynamics of the system are defined by

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

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 \( p_g \). If the agent crosses the line \( y = 0 \), we check whether its \( x \)-position is within the interval \([p_g - 0.5w_g, p_g + 0.5w_g]\). If this is not the case, we stop the episode as the agent has crashed into the wall. Each episode is terminated after a maximum of 100 steps. The reward function is given by

\[
r(s, a) = \exp (-0.6\|o - [x, y]\|_2),
\]

where \( o = [0, -3] \), \( \| \cdot \|_2 \) is the L2-Norm. The agent is always initialized at state \( s_0 = [0, 0, 3, 0] \).

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 \( \sigma_{\text{LB}} \) and \( D_{\text{KL,LB}} \) 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 for approximating the value function during policy optimization. For the allowed KL divergence, we observed best results when using \( \epsilon = 0.5 \) for the weight computation of the samples, but using a lower value of \( \epsilon = 0.2 \) when fitting the parametric policy to these weighted samples. We suppose that the higher value of \( \epsilon \) during weight computation counteracts the effect of the buffer containing policy samples from earlier iterations.

Looking at Figure 11, 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 \( s \in \mathbb{R}^{29} \) 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 \( a \in \mathbb{R}^{8} \) is defined by the 8 torques that are applied to the ant’s joints.

The context \( c = [p_g \, w_g] \in [-10, 10] \times [3, 13] \subset \mathbb{R}^{2} \) 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

\[
r(s, a) = 1 + 5 \exp \left( -0.5 \min(0, c_x - 4.5)^2 \right) - 0.3 \| a \|^2.
\]

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 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.
Figure 11: 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.

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 12, 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 $a \in \mathbb{R}^5$ are defined as the displacement of the current desired joint position. The state $s \in \mathbb{R}^{21}$ consists of the positions and velocities of the controlled joints, their current desired positions, the current three-dimensional ball position and its linear velocity.

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

| Environment         | $\mu_{\text{INIT}}$ | $\delta_{\text{INIT}}$ | $\mu_{\text{TARGET}}$ | $\delta_{\text{TARGET}}$ |
|---------------------|----------------------|-------------------------|------------------------|--------------------------|
| 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       | $[0.68 \ 0.9 \ 0.85]$ | $[0.03 \ 0.03 \ 0.3]$  | $[1.06 \ 0.85 \ 2.375]$ | $[0.8 \ 0.38 \ 1]$      |
Figure 13: 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.

As previously mentioned, the reward function is sparse,

\[ r(s, a) = 0.275 - 0.005\|a\|^2 + \begin{cases} 
50 + 25(n_s \cdot v_b)^5, & \text{if ball catched} \\
0, & \text{else}
\end{cases}, \]

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, \( n_s \) is a normal vector of the end effector surface and \( v_b \) 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.
\[
p_{\text{des}} = [0 -r \cos(\phi) 0.75 + r \sin(\phi)].
\]
Furthermore, the context determines the distance in \( x \)-dimension from which the ball is thrown
\[
p_{\text{init}} = [d_x \, d_y \, d_z],
\]
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 \( p_{\text{des}} \) 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 13 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. *ICLR*, 2018.
Filippo Aluffi-Pentini, Valerio Parisi, and Francesco Zirilli. The use of “continuous method” in complementarity problems. *IFAC Proceedings Volumes*, 18(2):273–276, 1985.
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 *NIPS*, 2017.
Adrien Baranes and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration for active motor learning in robots: A case study. In *IROS*, 2010.
Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In NIPS, 2016.

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

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.

J Bongard and Hod Lipson. Once more unto the breach: Co-evolving a robot and its simulator. In 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.

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 ICDL, 2008.

Hehe Fan, Liang Zheng, Chenggang Yan, and Yi Yang. Unsupervised person re-identification: Clustering and fine-tuning. 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 NeurIPS, 2019.

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

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In 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, pages 163–185, 1998.

Matthew M. Graham and Amos J. Storkey. Continuously tempered hamiltonian monte carlo. 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 ICML, 2018.
Nikolaus Hansen, Sibylle D Müller, and Petros Kountourisios. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

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, Szynon 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 *NIPS*, 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 *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 MM*, pages 547–556, 2014a.

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

Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G Hauptmann. Self-paced curriculum learning. In *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 *NIPS*, 2009.

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

Andras Gabor Kupcsik, Marc Peter Deisenroth, Jan Peters, and Gerhard Neumann. Data-efficient generalization of robot skills with contextual policy search. In *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. *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. *ICML*, 2019.

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

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

Stephan Mandt, James McInerney, Farhan Abrol, Rajesh Ranganath, and David Blei. Variational tempering. In *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 *ALT*, 2018.

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

Gerhard Neumann. Variational inference for policy search in changing situations. In *ICML*, 2011.

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

Pierre-Yves Oudeyer, Frédric 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. *TKDE*, 22(10):1345–1359, 2009.

Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Probabilistic movement primitives. In *NIPS*, 2013.

Simone Parisi, Hany Abdulsamad, Alexandros Paraschos, Christian Daniel, and Jan Peters. Reinforcement learning vs human programming in tetherball robot games. In *IROS*, 2015.
Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *AAAI*, 2010.

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

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 *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. *ICML*, 2018.

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

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

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *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 *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 *CVPR*, pages 2379–2386, 2013.

Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 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 *NIPS*, 2017.
Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *JMLR*, 10(7), 2009.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

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

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

Herke Van Hoof, Gerhard Neumann, Jan Peters, et al. Non-parametric policy search with limited information loss. *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.

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

Jan Wöhlke, Felix Schmitt, and Herke van Hoof. A performance-based start state curriculum framework for reinforcement learning. In *AAMAS*, pages 1503–1511, 2020.

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