Self-Paced Absolute Learning Progress as a Regularized Approach to Curriculum Learning

Ulla Scheler [* 1]   Tobias F. Niehues [* 1]   Pascal Klink [1]

Abstract

The usability of Reinforcement Learning is restricted by the large computation times it requires. Curriculum Reinforcement Learning speeds up learning by defining a helpful order in which an agent encounters tasks, i.e. from simple to hard. Curricula based on Absolute Learning Progress (ALP) have proven successful in different environments, but waste computation on repeating already learned behaviour in new tasks. We solve this problem by introducing a new regularization method based on Self-Paced (Deep) Learning, called Self-Paced Absolute Learning Progress (SPALP). We evaluate our method in three different environments. Our method achieves performance comparable to original ALP in all cases, and reaches it quicker than ALP in two of them. We illustrate possibilities to further improve the efficiency and performance of SPALP.

1. Introduction

In Reinforcement Learning (RL) an agent adapts its actions to maximize a reward signal it receives from its environment, e.g. (Sutton & Barto, 2018). In combination with methods from Deep Learning, RL was successfully employed in virtual gaming environments like Atari (Mnih et al., 2013) or Starcraft (Vinyals et al., 2019), as well as in physical manipulation tasks, e.g. (Andrychowicz et al., 2020; Rajeswaran et al., 2018; Kalashnikov et al., 2018). Before these levels of performance are reached, though, an agent often takes millions of steps while it tries different actions (this is called 'exploration') and encounters different variants of the environment. Instead of optimizing the agent's exploration of the parameter space, Curriculum Learning (CL) (Bengio

et al., 2009; Matiisen et al., 2020) tries to systematically present the agent with an optimal order of the parameter space samples. In this context, the algorithm that chooses the next training environment based on distributions generated by itself is often referred to as the teacher, while the executing agent is correspondingly called student.

There are several approaches to implementing a curriculum, for example deriving goal tasks via GANs (Florensa et al., 2018), incrementally adjusting the sampling distribution to resemble the distribution of a goal task, known as Self-Paced Deep Reinforcement Learning (SPDL) (Klink et al., 2020b), and sampling techniques that sample the parameter spaced based on recent learning advancement as in Absolute Learning Progress (ALP) (Portelas et al., 2020).

In this work, we apply insights from Self-Paced Learning as regularization in ALP settings and evaluate this approach in various environments.

1.1. Self-Paced Learning

The usual goal of self-paced learning (Klink et al., 2020b;a) is to master a goal task, defined as a subspace of the full parameter space, by gradually adjusting a distribution of tasks $p(c|\nu)$ used for sampling the training contexts c and its parameters ν to the goal task distribution $\mu(c)$. This can be accomplished by incorporating the KL-divergence between the sampling and the goal distribution into the objective function of the expected reward (1), given the reward function $R(\omega, c)$ and the policy π_ω, parameterized by ω. The parameter α controls the weight of the KL-divergence relative to the reward objective.

$$\max_{\omega, \nu} \mathbb{E}_{p(c|\nu)}[R(\omega, c)] - \alpha D_{KL}(p(c|\nu) \parallel \mu(c)) \quad (1)$$

Equation (1) can be solved by alternately maximizing w.r.t. ω and ν while gradually increasing α to converge to the goal distribution. In other words, the teacher attempts for the student to receive a good reward, while also exposing it to ever closer approximations of the goal environment.

Recent work in self-paced learning (Klink et al., 2021) has found that optimizing this modified objective function is equal to maximizing objective (2) w.r.t the sampling parameters ω.

*Equal Contribution [1]Department of Intelligent Autonomous Systems, Technical University of Darmstadt, Darmstadt, Germany. Correspondence to: Ulla Scheler <ulla.scheler@stud.tu-darmstadt.de>, Tobias F. Niehues <niehues.tobias@gmail.com>. This work originated from the master's module *Integrated Project: Learning Robots* at Technical University Darmstadt and we received course credit for it.

Please note that this version was originally derived from a supervised learning scenario in which the objective function aimed to minimize losses (Kumar et al., 2010; Jiang et al., 2015). Since application to an RL setting involves transforming the objective function from a loss minimization to a reward maximization and all losses previously were required to be all positive, the rewards fed into (2) now need to be all negative, e.g. by an appropriate transformation beforehand.

$$\max_{\omega} \mathbb{E}_{\mu(c)}[-\alpha(1 - \exp(\frac{1}{\alpha}R(\omega, c)))] \quad (2)$$

Note that SPDL only indirectly includes information about past experiences and learning progress in form of the learner's policy and the teacher's sampling distribution: The learner prefers to repeat behavior that previously performed well while the teacher fits its sampling distribution taking into account the rewards from previous iterations. In our approach we will combine SPDL's regularization objective with a more explicit take on learning progress and past experience.

1.2. Absolute Learning Progress (ALP)

The Absolute Learning Progress model (ALP-GMM) proposed by Portelas et al. (2020) is an approach to model intrinsic motivation in reinforcement learning (Oudeyer et al., 2007; Baranes & Oudeyer, 2013), which is inspired by models of intrinsic motivation in humans (Moulin-Frier et al., 2014). The term 'GMM' refers to the use of Gaussian Mixture models used to represent the sampling distribution. ALP-GMM uses the so-called Absolute Learning Progress as a measure of how much progress was gained by training on the current task (3). Tasks and their ALPs are saved into a reward history. The reward history is implemented as a first-in-first-out buffer of fixed size. To calculate the ALP for a reward r_{new} for a task θ_{new} we look up the nearest neighbor θ_{NN} in the reward history and use its reward r_{NN}:

$$\text{ALP} = |r_{new} - r_{NN}| \quad (3)$$

The sampling distribution during training is defined by a Gaussian mixture model (GMM) fitted to the ALPs of the batch of the last N sampled tasks. The next task is then with a probability p either sampled from the GMM or uniformly from the parameter space, to ensure exploration. A basic overview of ALP-GMM can be found in Figure 2. By basing most of the sampling on ALPs, the agent is provided with samples from subspaces of the parameter space that yielded the most progress in the last iterations. At the same time, the algorithm remembers the progress from older examples in form of the reward history.

One of the biggest advantages of ALP-GMM over other

Curriculum Learning approaches is its quite simple concept, for it does not need any structures like Markov Decision Processes (MDPs) or multi-armed bandits, as used by Matiisen et al. (2020) or Graves et al. (2017). Furthermore, it only needs tuning of relatively few hyperparameters. One must basically only provide hyperparameters defining how to fit the GMM, as the ALP metric itself does not need any additional hyperparameters.

Yet, please note that ALP-GMM does not differentiate whether the progress was achieved in a task where the agent is already quite proficient or in one where its performance was previously low. It cares only about the absolute difference between the current performance and the closest previous performance saved in the reward history. Thus ALP-GMM might waste computation on repeating already learned behaviour in new tasks or tasks it has not seen in a while. In these cases the ALP might be high, but does not signify any real learning progress.

The regularizer we introduce will extend ALP by mostly preferring environment parameterizations which yield higher rewards. The logic behind this is the following: Progress in a task from a lower to a low reward might not mean progress in the policy, i.e. the student most likely only utilizes what it already learned somewhere else but is not improving its overall behavior. In contrast, progress in a task from a high reward r_i to a higher reward r_{i+1} is more likely to stem from actual new learning as simple behaviors were probably already used to reach r_i and will not lead to further improvements.
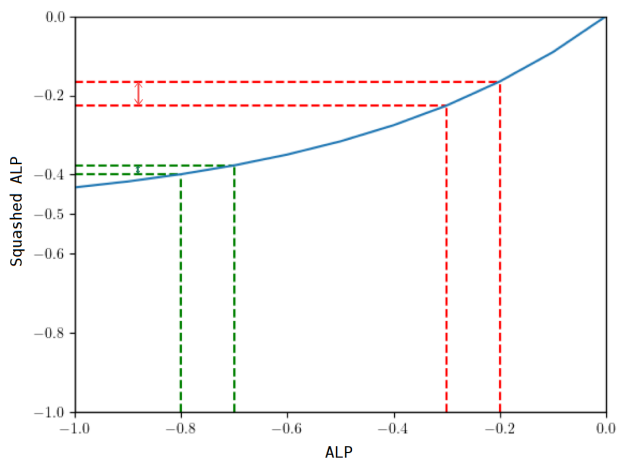

Figure 1. **Effect of squashing function $f_{\alpha}(x)$ on ALP calculation in SPALP**: Higher rewards are squashed less and thus the same absolute difference in rewards leads to a larger overall ALP. Squashing function plotted with $\alpha = 0.5$.
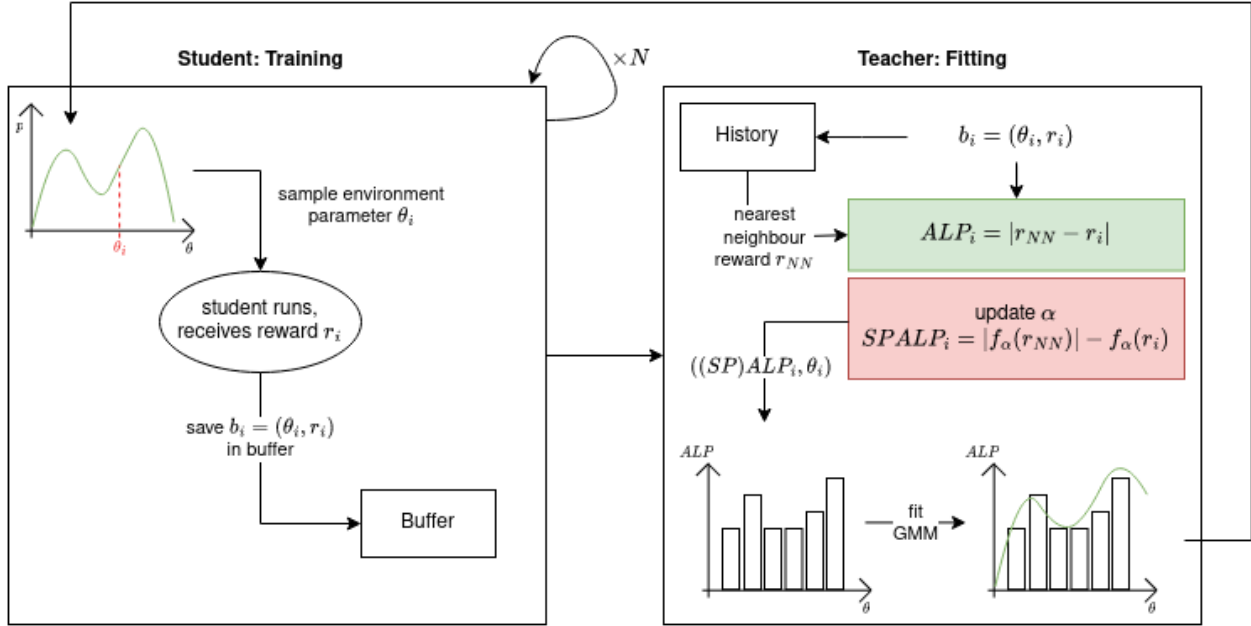
Figure 2. **A high-level illustration of how ALP-GMM and SPALP work**. Note that the only difference between the two algorithms lies in how the ALP are calculated. Left side: The student makes N runs. Environments are sampled from current teacher distribution each run. Right side: Calculate ALP for each parameter-reward-pair b_i in Buffer and fit GMM.

2. Self-Paced Absolute Learning Progress

We introduce Self-Paced Absolute Learning Progress (SPALP) as a variant of the ALP-GMM model, making use of the regularization in the modified objective function (2) from SPDL. This yields a regularization function for the rewards used when calculating the absolute learning progress given by (5), and leads to our so-called regularized ALP (4). An overview of our approach can be found in Algorithm 1 and Figure 2.

$$\text{ALP} = |f_\alpha(r_{\text{new}}) - f_\alpha(r_{\text{old}})| \tag{4}$$

We modify all rewards received from the different environments in two ways: Firstly, as ranges of rewards differ between environments, we normalize all environments to the interval $[0, 1]$. Secondly, since the regularization requires all rewards to be negative (as mentioned in 1.1), we shift these normalized rewards by -1 into the interval $I_r = [-1; 0]$ before applying the regularization.

$$f_\alpha(x) = -\alpha(1 - \exp(\frac{x}{\alpha})) \tag{5}$$

The function $f_\alpha(x)$, shown in Figure 3, can be thought of as a softener on ALPs calculated from low rewards. An example of this can be seen in Figure 1. Smaller values of α lead to more squashing. Rewards are squashed into the interval $[-\alpha, 0.0]$ instead of the regular interval I_r. Low rewards are squashed more strongly than high rewards. Thereby the ALPs calculated from low squashed rewards also become smaller. High rewards remain mostly untouched, because f_α approaches linear behavior near the origin. Thus, the regularization employs a rather conservative sampling strategy by focusing on tasks that already yield high rewards.

For high values of α the regularization function f_α converges to the identity mapping in I_r and we can think of the regularization as being turned off in this case.

For determining the exact value of α and thus the strength of the squashing in the regularization function f_α, we needed an additional condition. Our solution is to choose α such that it fulfills equation (6). That is, we choose α in a way that the squashed rewards are reaching a bound r_b, which is a new hyperparameter specific to SPALP. The bound r_b can intuitively be understood as the mean reward we are striving for in the region of the parameter space previously sampled from. If the rewards r_i were low, we need a stronger regularization to reach r_b, resulting in a low value of α and vice versa.
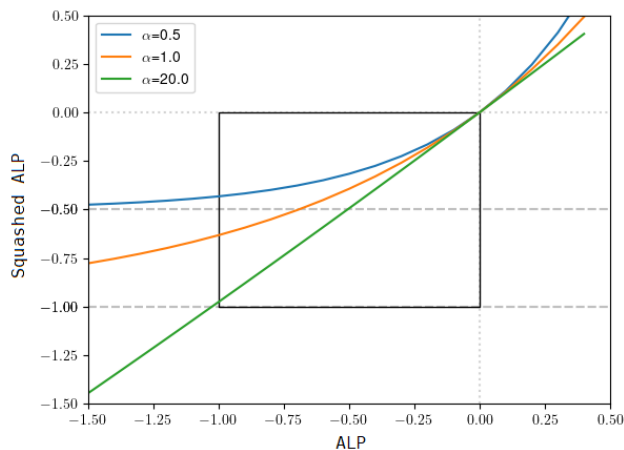
Figure 3. **Squashing function** $f_\alpha(x)$ for different values of alpha. In negative direction each curve converges to the negative α-values (indicated by the dashed lines), while they increase exponentially in the positive direction. Since all rewards are normalized into $[-1.0; 0.0]$, this is the interval of actual relevance for our work. Higher values of α lead to a more linear behavior, while smaller values lead to a stronger squashing of smaller rewards.

$$\frac{1}{N} \sum_i f_\alpha(r_i) = r_b \qquad (6)$$

By choosing α the way we do, the regularization gets weaker the closer the mean reward gets to the bound r_b, i.e. in general the longer the agent trains, the weaker the regularization becomes. To further support this behavior, we turn the regularization off completely if the mean reward of the last N samples exceeded r_b and fall back to vanilla ALP-GMM. This ensures exploration, since the regularization introduces a bias by focusing on tasks on which we were already performing well, while the unregularized version treats all ALPs equally (cf. figure 1). Once the unregularized mean reward falls below r_b, we turn it back on again.

The complete implementation is shown by Algorithm 1. Please note that we are not only sampling from the fitted GMM, but also uniformly take random samples from the parameter space with a probability of $p = 0.2$ to ensure exploration independent of the GMM, just as vanilla ALP-GMM does.

3. Experiments

For evaluation and comparison of our teaching algorithm we used three different environments as shown in Appendix A; a toy environment where the progress of rewards in the sample space is approximated by a mathematical function for different subspaces, the bipedal walker environment

---

Algorithm 1 Self-Paced Absolute Learning Progress

1: **Input:** fitting rate N, bounded parameter space $\mathcal{P}$, student $\mathcal{S}$, teacher $\mathcal{T}$
2: **Initialize** First-in-First-Out buffer $\mathcal{B}$ with max size N
3: **Initialize** reward history database $\mathcal{H}$
4: **Bootstrapping** for N iterations
5: **while** not terminated **do**
6:     update α
7:     calculate ALP of last N steps using the reward of the nearest neighbor (in parameter space) in H
8:     fit teacher-GMM on ALPs
9:     **for** N runs **do**
10:         sample θ_i from $\mathcal{P}$ using the GMM of $\mathcal{T}$
11:         write reward r_i from $\mathcal{S}$ in environment E_θ into B
12:     **end for**
13: **end while**
14: **return** $\mathcal{S}$

---

as used by Portelas et al. (2020) for evaluation of ALP-GMM and the ball catching environment as used by Klink et al. (2020b; 2021) for evaluation of their Self-Paced Deep Reinforcement Learning.

3.1. Toy Environment

The toy environment already provided by Portelas et al. (2020) subdivides an n-dimensional space into (hyper)cubes along each dimension. To simulate in making progress on the rewards, sampling from a cube linearly increases the reward for sampling from this very cube the next time. At first, only the cube in the bottom left (equivalent to the supposedly easiest parameter configurations) yields a reward when being sampled from. All other cubes need to be unlocked first by having a 'mastered' hypercube in their immediate vicinity. A cube is 'mastered' if it yielded a reward higher than 75. The maximum reward for each cube is capped at 100.

We extended this environment by incorporating additional reward progress shapes besides the aforementioned linear behavior. To achieve a more realistic change in the rewards, we found a sigmoidal curve to be an adequate choice, as the reward needs some time at first to increase more strongly (equivalent to an agent getting used to a task for the first time) and only yields small progress before saturating to the reward maximum (equivalent to the agent possibly not being able to make any more progress).

Further, we wanted to also add a concept of transfer learning, where newly discovered cubes benefit from performances in neighboring cubes in their direct vicinity. For this, when sampling from each cube for the first time, it already yields a reward equal to the average reward already obtained from its neighboring cubes.

Our main metric used for evaluation of different algorithms

in this environment was the percentage of 'mastered' hypercubes over the whole parameter space. Table 1 shows this performance in comparison to random sampling and ALP-GMM.

$r_b = -0.4$ resulted in the best performance remarkably often; still, this is not in line with the results of the other environments, shown in sections 3.2 and 3.3.
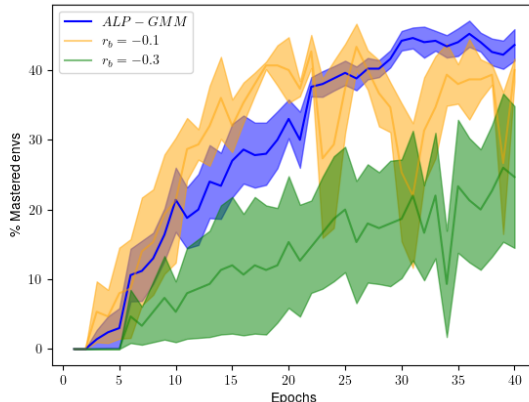
Table 1. **Performance in the toy environment for different configurations** ($\pm$ standard error). 'TL' indicates whether Transfer Learning was simulated. Performance measured as percentage of mastered hypercubes. For $d = 2$, each dimension consisted of 10 cubes; 5 cubes per dimension for $d = 3$. 3 runs per configuration with 40000 episodes each. For SPALP the best performing r_b was chosen each time. Performances of the best teachers per configuration are highlighted bold.

d	REWARD	TL	TEACHER	r_b	PERFORMANCE
2	LINEAR		RANDOM	-	14.67 ± 0.17
		$\times$	ALPGMM	-	49.89 ± 1.47
			SPALP	-0.3	$\mathbf{53.00 \pm 7.51}$
			RANDOM	-	30.70 ± 0.37
		$\checkmark$	ALPGMM	-	80.86 ± 2.29
			SPALP	-0.4	$\mathbf{91.33 \pm 2.19}$
	SIGMOID		RANDOM	-	26.96 ± 0.29
		$\times$	ALPGMM	-	43.41 ± 1.62
			SPALP	-0.4	$\mathbf{52.33 \pm 6.69}$
			RANDOM	-	58.63 ± 0.57
		$\checkmark$	ALPGMM	-	85.26 ± 1.47
			SPALP	-0.5	$\mathbf{91.00 \pm 3.51}$
3	LINEAR		RANDOM	-	15.47 ± 0.15
		$\times$	ALPGMM	-	47.79 ± 2.98
			SPALP	-0.4	$\mathbf{65.07 \pm 1.48}$
			RANDOM	-	57.69 ± 1.51
		$\checkmark$	ALPGMM	-	96.95 ± 1.56
			SPALP	-0.4	$\mathbf{100.00 \pm 0.00}$
	SIGMOID		RANDOM	-	33.24 ± 0.79
		$\times$	ALPGMM	-	42.79 ± 1.87
			SPALP	-0.9	$\mathbf{52.00 \pm 7.43}$
			RANDOM	-	99.82 ± 0.08
		$\checkmark$	ALPGMM	-	92.83 ± 2.17
			SPALP	-0.6	$\mathbf{100.00 \pm 0.00}$

3.2. Bipedal Walker Environment

We use the Bipedal Walker environment by Portelas et al. (2020) in the standard configuration provided by the author (default leg length of the walker, obstacle spacing from 0 to 6 and obstacle height from 0 to 3), which is in turn adapted from OpenAI (2018).

The goal of the agent in this environment is to walk as long as possible without falling down. Reward is received for moving forward, while the application of motor torque costs a small amount of points and falling down is penalized and ends the episode. Falling is caused either by the walker's inability to walk or inability to cross an obstacle. Note that the parameter space, consisting of obstacle spacing and

Figure 4. **Percentage of mastered environments for the bipedal walker environment**. After each training epoch the student is tested in 50 environments of the parameter space. An environment counts as "mastered" if the student achieves more than a certain reward. Figure shows mean performance and standard error of 10 (ALP-GMM) or 3 (SPALP) seeds.

obstacle height, contains regions that are infeasible due to the agent's leg length. The state of the agent consists of several velocity measures of the joints, 10 lidar rangefinder measurements and legs contact with the ground.

As student we used SAC (Haarnoja et al., 2018) from the `Stable Baselines` library (Hill et al., 2018), again slightly altered by (Portelas et al., 2020). For comparison, ALP-GMM was always run with the parameters provided by Portelas et al. (2020).

We evaluated the performance of the teachers by evaluating how many of 50 different walker environments they were able to master. Because these test environments were chosen at random from all over the parameter space, i.e. with varying obstacle spacing and obstacle height, there are some that were infeasible to the default walker due to its leg length. An environment counts as "mastered" if the student reached more than 230 points. This arbitrary bound was again modeled after (Portelas et al., 2020).

The results are shown in figure 4. Please note that due to time and computing constraints, the SPALP results represent only three seeds and correspondingly more noise. To illustrate: The dips in the SPALP curves are due to noise of an errant seed. SPALP with $r_b = -0.1$ reaches an overall performance similar to ALP-GMM but does so earlier. In the case of $r_b = -0.1$, the reward bound was not reached in any of the seeds and correspondingly, squashing was always turned on. To show the negative influence of a badly

chosen reward bound, we also included the performance of the worst bound ($r_b = -0.3$). We should note that low r_b ($-0.8; -0.9$) also performed well in the bipedal walker environment. In these cases the reward bound was reached before the first fitting of the teacher distribution, though. As this is equivalent to running vanilla ALP-GMM, we did not include these runs in the results.

To study the differences in the teacher's sampling distributions, we plotted the Gaussian Mixture Models after each fitting period. Across all the runs we analyzed, it became apparent that SPALP focuses on the easiest subspace of the parameter space, while ALP advances to more difficult regions more quickly (moving the concentration of samples diagonally to the bottom right in the parameter space). This is shown exemplarily for two successful seeds in figure 5. In the case of unsuccessful SPALP seeds, we noticed that from the beginning the teacher did not focus on the easiest subspace to the same extent.
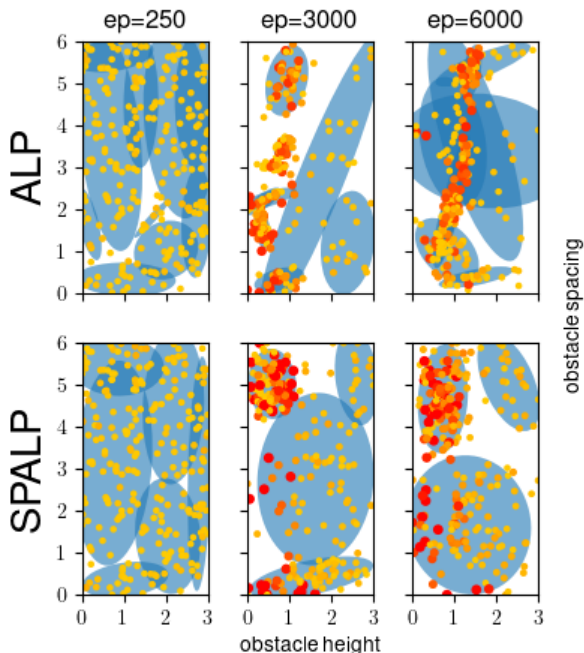

Figure 5. **Different sampling behaviors of ALP-GMM and SPALP**, illustrated in the bipedal walker environment for episodes 250, 3000 and 6000. Blue ellipses represent the GMMs, the dots represent single samples. Yellow samples represent a low (regularized) ALP, red samples represent a high (regularized) ALP. Both algorithms start with a general exploration period of 250 episodes. After that, SPALP focuses its sampling on the easiest subspace of the parameter space (low and widely spaced obstacles, upper left corner). In contrast, ALP-GMM samples more broadly in the low obstacle height part of the parameter space. Although the data represents a single, successful seed per algorithm of ALP-GMM and SPALP with $r_b = -0.1$, the patterns were consistent over all successful seeds.

3.3. Ball Catching Environment

The ball catching environment is the same as used by Klink et al. (2020b; 2021); consult his paper for any experimental details on the simulation please, since for our purposes not all simulation details are relevant[1].

The environment is defined by the distance from which the ball is thrown and its target position in the catching plane of the robot. The reward function of this task is sparse, as the agent only receives a reward when catching the ball but control cost for any movements made at all times.

To provide the robot with a good initial behavior and the possibility to focus on the catching task instead of learning its plain controls, it was pre-trained to a policy by which it is capable of holding its initial position.

For training we used SAC (Haarnoja et al., 2018) as the student, as provided by the `Stable Baselines` library (Hill et al., 2018) and used by Klink et al. (2020b; 2021). The GMMs in ALP and SPALP were fitted after each 200 iterations during training.

We then evaluated the performance of the trained model for different values of r_b in SPALP by running 200 runs on random ball catching tasks and measuring the percentage of successful catches, exemplary shown in Figure 7. The value of $r_b = -0.1$ was one of the best performers while $r_b = -0.7$ was one of the worst. In general, high r_b ($-0.1; -0.2$) and low r_b ($-0.8; -0.9$) performed well in the ball catching environment. Yet, for low r_b the bound was often reached almost instantly, so the regularization was never put to work in the first place - for this reason we excluded those results.

In contrast to the bipedal walker environment (see section 3.2), we did not find any prominent differences in the ALP and SPALP teachers' sampling behaviors for the ball catching environment. Samples yielding a high ALP were spread over a large section of the parameter space.

We also evaluated how the value of α changed during training runs. Low values of α lead to strong squashing, which is only needed in order to fulfill equation (6) when the mean reward is far from the bound r_b. Because of this we supposed that high values of α (and thus a weak regularization) should occur shortly before surpassing r_b; cf. figure 3 for more intuition on that. Our findings support that hypothesis, as figure 6 shows.

4. Discussion

4.1. Overall performance of SPALP

The results from the experiments show that SPALP with a properly chosen reward bound can reach at least the same performance levels as vanilla ALP-GMM but might do so

---

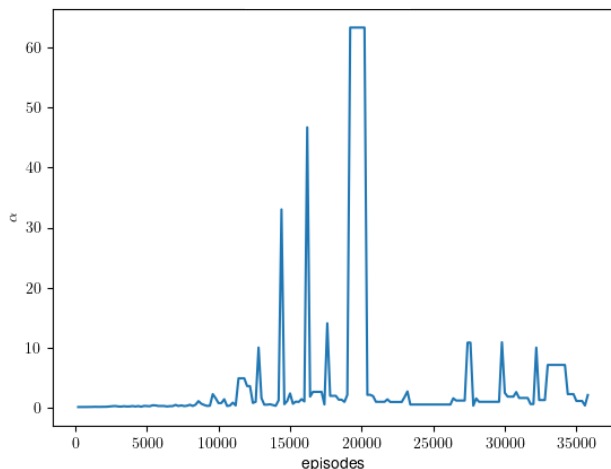[1]For simulation the MuJoCo physics engine was used.

Figure 6. **Evolution of** α for one sample run in the ball catching environment with hyperparameter $r_b = -0.1$. Most of the time the value stays below 1, but is spiking to really high values every now and then. These spikes can be explained by choosing a new α when the mean reward is already close to r_b. Since we choose α s. t. the mean regularized reward is equal to r_b - cf. equation (6) -, this leads to a high α and thus a weak regularization. This is also supported by spikes regularly occuring before plateaus. The plateaus indicate every time the bound r_b was surpassed by the mean reward and we fall back to vanilla ALP-GMM, thereby also turning off any α-updates.

Figure 7. **Percentage of successful catches of 200 throws in the ball catching environment** after training with ALP-GMM and SPALP (different r_b). The figure shows mean performance and standard error for five seeds per parameter value.

sooner. This is exactly what we would expect from a reward regularizer: SPALP uses the same semantic structures as ALP-GMM, so it seems unreasonable to expect it to surpass the performance of ALP-GMM in absolute terms.

We generally expect SPALP to perform even better, i.e. quicker, than we were able to show so far. This is mostly due to two facts: First, we only performed little tuning on the parameters. Except for the reward bound, all parameters were chosen as in (Portelas et al., 2020). We expect tuning to be especially beneficial for the size of the reward history and the length of the initial exploration period and explain this reasoning below. Second, SPALP performed best in the walker and the ball catching environment with a reward bound r_b of -0.1. This value lies on the edge of the parameter space we tried during tuning, while the general tendency of the tuning was that higher bounds performed better. Performance will thus probably improve further with a higher reward bound, e.g. $r_b = -0.05$ or even $r_b = 0.1$.

4.2. A more nuanced approach to squashing

Picking one reward bound as the only hyperparameter for how much we squash might be too crude a heuristic. The advantage of SPALP comes from focusing on an easier subspace first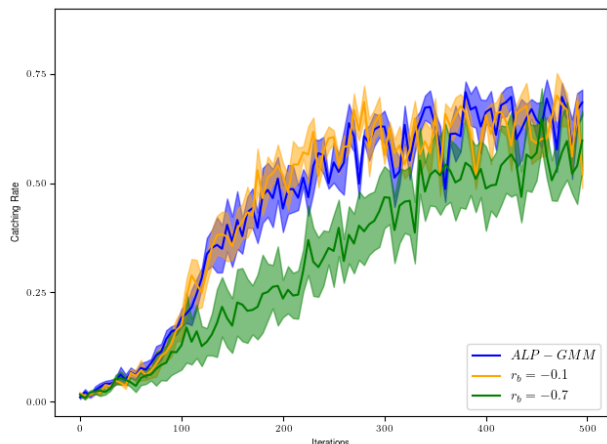, and squashing is strongest the further away the student is from the reward bound. As it gets closer during training, the squashing gets weaker, allowing the student to explore environments where the overall reward is smaller, while still taking into account the learning progress, i.e. the relative reward differences.

In the current implementation of SPALP, as soon as the student falls below the reward bound, squashing is turned on again and the teacher again favors parameterizations of the environment that yield high rewards. If this happens to be the easiest part of the subspace where we already sampled, the agent cannot learn anything new. In this case, we enter a cycle in which the student trains on easy environments until the reward bound gets turned off, then explores for a short while until it has to train on the easy environments again. This turning-off-and-on-again pattern is indeed what we saw in case of the ball catching environment (see figure 6). This behavior is problematic: if progress can no longer be made in the easy subspace, but only in parts of the parameter space where the tasks are harder, learning is slower and rewards are thus smaller for a longer time.

One solution might consist of lowering the reward bound each time it is reached, like a ladder, thereby lowering squashing overall and lessening the sampling disadvantage of lower reward subspaces.

Note that whether we encounter the on-off-problem is also influenced by the size of the reward history, upon which the ALP are calculated (see section 2): If the reward history is long enough, the teacher can remember that the student did not make sizable learning progress in the easy subspace before squashing was turned off. In this case, even if squashing is turned on again, the teacher still favors the more difficult subspaces, because squashing can only make a small ALP

of a high-reward environment bigger to some extent.

It follows that another solution might be to change the implementation of the reward history. We could simply make the FIFO buffer bigger. Or we could change it from one single FIFO buffer for the complete parameter space to several smaller buffers that are designated for distinct parts of the parameter space. In the latter case, the teacher remembers how much progress was made in one area until we sample there again. The last solution would be more principled but would also complicate the implementation.

Future work might try these more nuanced approaches on its own or in combination.

4.3. Stronger squashing might require longer initial exploration

Any advantage of the SPALP teacher over the ALP teacher is rooted in a different distribution from which it draws the samples it presents to the student. To study this, we explored the sampling behavior of the SPALP teacher in the bipedal walker environment in section 3.2. We found that in badly performing seeds there was a pattern that the teacher did not focus as strongly on the easy subspaces of the parameter space. Typically, this happened when the very first fitting of the GMM did not see a lot of samples in this region. While this is only one reason why a seed might perform badly, it is within our control to change it and thus deserving of further analysis.

When squashing leads to higher ALP on high-reward tasks, the GMMs concentrate most of their probability mass on a few tasks. If the initial exploration phase is too short and the teacher did not sample from the easy regions, the GMMs are fitted to the wrong tasks. In this case, the teacher can only recover through the small amounts of samples dedicated to random exploration. Thus, we should think of high squashing as something we do only after we have some confidence that our previous samples overall came from the right spot in the parameter space. One simple way to make this more probable is to increase the period of initial exploration before the first GMM is fitted.

4.4. Differences in the SPALP performance over environments

In the case of the walker environment, we found differences in how the ALP-GMM and SPALP teachers sample: In well performing seeds, SPALP focused on the easiest part of the parameter space. In the case of the ball catching environment, we did not find any differences. This goes hand in hand with the overall performance: In the walker environment, SPALP was able to learn quicker than ALP-GMM. In the ball catching environment, the results of SPALP and ALP-GMM were very similar to each other.

Regarding the ball catching environment, we further remarked that the samples yielding high ALP were mostly widespread over the parameter space of the ball catching environment. In this case, there is no region SPALP can focus on.

It bears repeating: Any advantage of the SPALP teacher over the ALP-GMM teacher is rooted in a different distribution from which it draws the environment parameterizations it presents to the student. If there is no region to focus on, we cannot expect a difference between ALP-GMM and SPALP. Consequently, we would expect no improvement in the performance of SPALP when using a higher reward bound: if there is no clear region where squashed ALPs are higher, more squashing won't help.

The lack of an apparent "easy" region might also explain the success of Klink et al. (2020b) over ALP-GMM in the ball catching environment, because in their approach they explicitly define a starting distribution over the parameter space for the teacher to focus on, which they then slowly expand. Results from the toy environment also indicate that the shape of an environment's reward function influence the effectiveness of SPALP. The toy environment is a handcrafted approach that might not be directly comparable to more realistic applications, like the bipedal walker and ball catching environment. Nevertheless, in this simple environment we saw significant differences in the advantage of SPALP over ALP depending on the reward function.

Future work could determine in which environments SPALP works better or worse than standard ALP.

5. Conclusion

We aimed to create an improved version (SPALP) of the already well-performing ALP-GMM from Portelas et al. (2020) by extending it with a new regularization method inspired by recent progress in Self-Paced Learning. A major short-coming of our work lies in the small amount of random seeds used for some evaluations. We evaluated our approach in three environments, one of which was a handcrafted approach. Only in the handcrafted approach was SPALP able to clearly beat ALP-GMM. We explained how these results might be replicated for the other two environments by stronger squashing, a longer exploration period and a longer memory of the teacher.

Our results indicate that it depends on the environment and its reward structure how much performance benefits from regularization. Future work could examine these differences in environments in a more principled way.
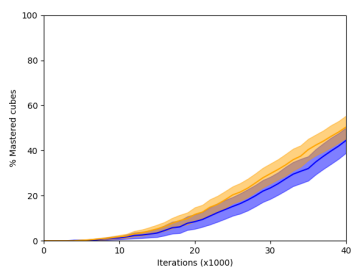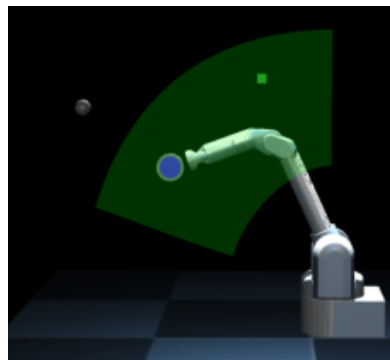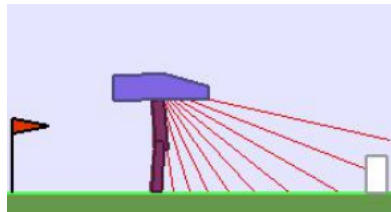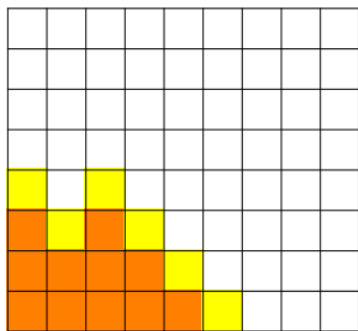
Acknowledgements

References

Andrychowicz, O. M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, January 2020. Publisher: SAGE Publications Ltd STM.

Baranes, A. and Oudeyer, P.-Y. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161.

Florensa, C., Held, D., Geng, X., and Abbeel, P. Automatic goal generation for reinforcement learning agents. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1515–1528. PMLR, 10–15 Jul 2018.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. Automated curriculum learning for neural networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1311–1320. PMLR, 06–11 Aug 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

Jiang, L., Meng, D., Zhao, Q., Shan, S., and Hauptmann, A. G. Self-paced curriculum learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., and Levine, S. Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. In *Conference on Robot Learning*, pp. 651–673, October 2018.

Klink, P., Abdulsamad, H., Belousov, B., and Peters, J. Self-paced contextual reinforcement learning. In Kaelbling, L. P., Kragic, D., and Sugiura, K. (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 513–529. PMLR, 30 Oct–01 Nov 2020a.

Klink, P., D'Eramo, C., Peters, J., and Pajarinen, J. Self-paced deep reinforcement learning. In *NeurIPS*, 2020b.

Klink, P., Abdulsamad, H., Belousov, B., D'Eramo, C., Peters, J., and Pajarinen, J. A probabilistic interpretation of self-paced learning with applications to reinforcement learning. *CoRR*, abs/2102.13176, 2021.

Kumar, M., Packer, B., and Koller, D. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23:1189–1197, 2010.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher–student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3732–3740, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013. arXiv: 1312.5602.

Moulin-Frier, C., Nguyen, S. M., and Oudeyer, P.-Y. Self-organization of early vocal development in infants and machines: the role of intrinsic motivation. *Frontiers in Psychology*, 4:1006, 2014.

OpenAI. Openai gym bipedal walker environment. https://github.com/openai/gym/blob/master/gym/envs/box2d/bipedal_walker.py, 2018.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

Portelas, R., Colas, C., Hofmann, K., and Oudeyer, P.-Y. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In Kaelbling, L. P., Kragic, D., and Sugiura, K. (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 835–853. PMLR, 30 Oct–01 Nov 2020.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. *arXiv:1709.10087 [cs]*, June 2018.
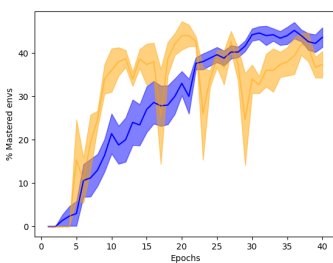
Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, November 2019. Number: 7782 Publisher: Nature Publishing Group.
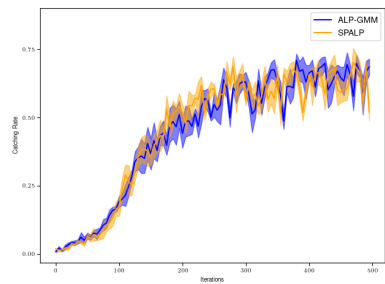
A. Environments Overview

(a) **Toy Environment**. Approximates a parameter space as a set of hypercubes that individually yield higher rewards proportionally to how often they were being sampled from.

$r_b = -0.4$ for SPALP. Originally introduced by Portelas et al. (2020).

(b) **Bipedal Walker Environment**. The bipedal walker has to learn to walk in a way to overcome obstacles of different heights and spacings in its way.

$r_b = -0.1$ for SPALP. Also used for evaluation of ALP-GMM by Portelas et al. (2020). Top image taken from figure 1a of the paper by Portelas et al. (2020).

(c) **Ball Catching Environment**. The catcher has to learn how to move in order to catch a ball thrown somewhere in the catching plane, indicated in green in the top figure.

$r_b = -0.1$ for SPALP. Used by Klink et al. (2020b) for evaluation of their Self-Paced Deep Learning. Top image taken from figure 1c of the paper by Klink et al. (2020b).

Figure 8. Renderings of the different environments and their respective best performance plots with ALP-GMM (blue) and SPALP (orange).