Deep Black-Box Reinforcement Learning with Movement Primitives

Anonymous Author(s)
Affiliation
Address
email

Abstract: *Episode-based reinforcement learning* (ERL) algorithms treat *reinforcement learning* (RL) as a black-box optimization problem where we learn to select a parameter vector of a controller, often represented as a movement primitive, for a given task descriptor called a context. ERL offers several distinct benefits in comparison to step-based RL. It generates smooth control trajectories, can handle non-Markovian reward definitions, and the resulting exploration in parameter space is well suited for solving sparse reward settings. Yet, the high dimensionality of the movement primitive parameters has so far hampered the effective use of deep RL methods. In this paper, we present a new algorithm for deep ERL. It is based on differentiable trust region layers, a successful on-policy deep RL algorithm. These layers allow us to specify trust regions for the policy update that are solved exactly for each state using convex optimization, which enables policies learning with the high precision required for the ERL. We compare our ERL algorithm to state-of-the-art step-based algorithms in many complex simulated robotic control tasks. In doing so, we investigate different reward formulations - dense, sparse, and non-Markovian. While step-based algorithms perform well only on dense rewards, ERL performs favorably on sparse and non-Markovian rewards. Moreover, our results show that the sparse and the non-Markovian rewards are also often better suited to define the desired behavior, allowing us to obtain considerably higher quality policies compared to step-based RL.

Keywords: Deep Movement Primitives, Deep Episode-Based RL, Differentiable Trust Region Layers

1 Introduction

Reinforcement learning (RL) problems can be viewed from a step-based [1, 2, 3] and an episode-based perspective [4, 5, 6]. In the former, most commonly found in deep RL, a policy selects an action for each state of the trajectory. Consequently, step-based RL requires Markovian reward definitions. Moreover, the exploration in action space often induces a random walk that inadequately explores the entire behavior space of the agent. As a result, most step-based RL algorithms only work well with dense rewards, where the agent receives a reward signal at each time step, rather than only at the last step. We refer to the latter as a sparse reward setting.

In the *episode-based RL* (ERL) perspective, we choose the complete behavior during the episode with respect to the controller parameters in the beginning of the episode. Typically, simple controllers are used that are valid only for executing a single trajectory, such as *movement primitives* (MPs). At the beginning of an episode, the MP parameters are adapted to a context vector that serves as task descriptor and may contain e. g. different initial joint configurations, the goal to reach, or start positions as well as object and obstacles locations. ERL allows for efficient exploration of the behavior space because exploration is implemented in the parameter space of the MP, allowing these algorithms to learn from sparse or even non-Markovian rewards. In addition, ERL inherently gen-

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

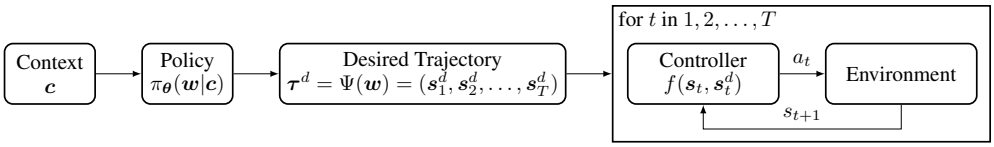

Figure 1: Overview of the proposed Black-Box Reinforcement Learning (BBRL) framework. The normal distributed policy predicts, given the context $\boldsymbol{c}$, the parameters of a movement primitive that translates to a desired trajectory τ^d. A trajectory tracking controller f generates low-level actions a_t given the current s_t and the desired state s_t^d from the trajectory τ^d.

erates smooth control trajectories due to the use of MPs, which often leads to more energy efficient behavior. However, ERL has so far not gained much popularity as it does not exploit the time-series structure of the RL problem and is therefore considered less data efficient, at least in the dense reward setting. So far, there is no deep learning method to adapt the MP parameters to the context, which limits ERL to linear adaptation strategies [7, 8].

This paper offers two main contributions: (i) We propose a new algorithm for deep ERL. The high-dimensional action space imposed by the MP parameters requires highly accurate action selection, hence robust and precise policy updates are needed. We adapt a recent step-based RL algorithm based on differentiable trust region projection layers (TRPLs) [9] to the ERL setting. Unlike methods such as *proximal policy optimization* (PPO) [2], that are based on approximated trust regions, TRPL offers exact trust regions for the policy updates which is essential for learning precise policies in high-dimensional action spaces. As second contribution, (ii) we show that ERL can be superior to step-based RL for tasks that are more easily define by sparse or non-Markovian rewards than by a dense reward. To this end, we have investigated different reward formulations for a variety of complex simulated robot control tasks. Our experiments show that step-based algorithms have difficulties learning high-quality policies in sparse or non-Markovian reward settings, while our proposed algorithm scales well to these reward settings. We further show that dense Markovian rewards can be very restrictive in terms of specifying a desired behavior in our domains. For example, if we want to achieve energy efficient, time-optimal reaching motions, sparse rewards can be used that only penalize the distance to the goal in the last time step. Non-Markovian rewards are useful for tasks with inherent non-Markovian optimality descriptors, e. g., the maximum height during a jump or the minimum distance of a bat to a ball. As a consequence, our algorithm can master tasks where step-based RL performs poorly even after intensive engineering of the used dense rewards.

In summary, our findings show that our algorithm has the following benefits: (a) it produces inherently smooth control policies, (b) offers efficient exploration in parameter space, which in turn enables learning with (c) sparse and non-Markovian rewards. These rewards also (d) allow for a more direct specification of the desired behavior, which leads to (e) higher quality of the learned behavior. These benefits come at the cost of an increased data complexity (a factor of 2 to 3).

2 Background and Related Work

Contextual Episode-based Reinforcement Learning. Contextual episode-based policy search [4, 8] treats RL as black-box optimization problem where a contextual search distribution $\pi(\boldsymbol{w}|\boldsymbol{c})$ over the controller parameters $\boldsymbol{w}$ is optimized to maximize the expected return $R(\boldsymbol{w}, \boldsymbol{c})$, i. e.,

$$\underset{\pi(\boldsymbol{w}|\boldsymbol{c})}{\arg\max}\, \mathbb{E}_{p(\boldsymbol{c})}\left[\mathbb{E}_{\pi(\boldsymbol{w}|\boldsymbol{c})}[R(\boldsymbol{w}, \boldsymbol{c})]\right],$$

where $p(\boldsymbol{c})$ denotes the context distribution given by the task. Due to the black-box nature of the problem, no structural assumptions are made for the return function $R(\boldsymbol{w}, \boldsymbol{c})$, i. e., the return can be any non-Markovian function of the resulting trajectory. Most ERL algorithms only consider the non-contextual setting, where different optimization techniques have been used, such as policy gradients [10], natural gradients [11], stochastic search strategies [12, 13, 14], or trust-region optimization techniques [5, 6, 8]. The only methods that incorporate context adaptation [8, 14] con-

sider a linear mapping from context to parameter space. We, in contrast, consider highly nonlinear context-parameter relationships using deep learning.

Step-Based Reinforcement Learning. Unlike episode-based methods, step-based approaches maximize the expected return by optimizing an action-selection policy that chooses a new action in each time step of the trajectory. While the goal remains to learn optimal trajectories, they interact directly with the environment based on raw actions, such as positions, velocities, or torques generated from the current state information. One such method is the trust region policy optimization algorithm [15], which first introduced trust region methods to deep RL, but is rather complex and difficult to scale. PPO [2] simplifies the update by introducing a clipping heuristic to the policy gradient objective. An alternative to the above policy gradient methods are policy iteration based approaches [16].

Differentiable Trust Region Layer. Differentiable trust region projection layers (TRPLs) [9] present a scalable and mathematically sound approach for enforcing trust regions in step-based deep RL. While PPO [2] is also motivated by trust-region updates of the policy, it cannot enforce the trust region exactly. TRPL provides an efficient way to enforce a trust region as well as more stability and control during training, while reducing the dependency on code level choices [17].

The layer takes the output of a standard Gaussian policy as input in terms of mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$ and projects it into the trust region if the given mean and variance violate their respective bounds. This projection is done for each input state individually. Subsequently, the projected Gaussian policy distribution with parameters $\tilde{\boldsymbol{\mu}}$, $\tilde{\boldsymbol{\Sigma}}$ is used for any further steps, e. g. for sampling and/or loss computation. Formally, the layer solves the following two optimization problems for each state $\boldsymbol{s}$

$$\arg\min_{\tilde{\boldsymbol{\mu}}_s} d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}(s)\right), \quad \text{s.t.} \quad d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}_{\text{old}}(s)\right) \leq \epsilon_{\boldsymbol{\mu}}, \quad \text{and}$$

$$\arg\min_{\tilde{\boldsymbol{\Sigma}}_s} d_{\text{cov}}\left(\tilde{\boldsymbol{\Sigma}}_s, \boldsymbol{\Sigma}(\boldsymbol{s})\right), \quad \text{s.t.} \quad d_{\text{cov}}\left(\tilde{\boldsymbol{\Sigma}}_s, \boldsymbol{\Sigma}_{\text{old}}(\boldsymbol{s})\right) \leq \epsilon_{\Sigma},$$

where $\tilde{\boldsymbol{\mu}}_s$ and $\tilde{\boldsymbol{\Sigma}}_s$ are the optimization variables for input state $\boldsymbol{s}$ and ϵ_μ and ϵ_Σ are the trust region bounds for mean and covariance, respectively. Finally, d_{mean} and d_{cov} are the similarity metrics for the mean and covariance of a decomposable distance or divergence measure. Otto et al. [9] proposed three such measures, we will, however, only leverage the decomposed KL-divergence in this work. The trust region for the KL-divergence measure can be made fully differentiable as shown in [9] and is also explained in more detail in Appendix A.

Reinforcement Learning with Movement Primitives. While most work on RL with movement primitives [5, 18, 19, 20] concentrates on learning a single MP parameter vector for a single task configuration, a few methods allow linear adaptation [6, 21, 22] of the MP's parameter vector to the context. In terms of step-based deep RL, Bahl et al. [23] propose *neural dynamic processes* (NDP). Their goal is to embed the structure of *dynamical movement primitives* (DMPs) into deep policies by reparametrizing action spaces via second-order differential equations. This can be seen as intersection between step-based and trajectory methods by learning subtrajectories via DMPs spanning multiple timesteps. Their approach allows for effective replanning, but unlike our approach, their main exploration still takes place at the action level rather than at the trajectory level. Hence, equivalent to standard step-based approaches, they have difficulties with sparse and non-Markovian rewards. Lastly, using DMPs requires several numerical integration steps that also need to be differentiated, which is computationally expensive.

3 Deep Black-Box Reinforcement Learning

Formally, we learn a policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{c})$ that models a distribution over the MP parameters $\boldsymbol{w}$ based on the context information $\boldsymbol{c}$. For most traditional RL tasks, the context space is often a subset of the observations space. As we learn a policy in the parameter space $\boldsymbol{w}$, exploration is also taking place at this level. Instead of sampling actions in each time step, we sample a parameter vector $\boldsymbol{w}$ only in the beginning of each episode. This leads to exploration at the level of desired trajectories and thus to much smoother and temporally correlated exploration. An overview is shown in Figure 1.

3.1 Episode-Based Reinforcement Learning Objective

Similar to step-based policy gradient [24, 15], we can also optimize the advantage function using the likelihood ratio gradient and an importance sampling estimator in the ERL setting. Our objective has thus the following form

$$\hat{J}(\pi_{\boldsymbol{\theta}}, \pi_{\boldsymbol{\theta}_{\text{old}}}) = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{c},\boldsymbol{w})\sim p(\boldsymbol{c}),\pi_{\boldsymbol{\theta}_{\text{old}}}} \left[\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{c})}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{w}|\boldsymbol{c})} A^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\boldsymbol{c}, \boldsymbol{w}) \right],$$

which is maximized w.r.t $\boldsymbol{\theta}$. The term $A^{\pi}(\boldsymbol{c}, \boldsymbol{w}) = \mathbb{E}\left[R|\boldsymbol{c}, \boldsymbol{w}; \pi\right] - \mathbb{E}\left[R|\boldsymbol{c}; \pi\right]$ denotes the advantage function and $\pi_{\boldsymbol{\theta}_{\text{old}}}$ is the old policy used for sampling. We can further make use of a learned context-value function $V_{\phi}(\boldsymbol{c}) \approx \mathbb{E}\left[R|\boldsymbol{c}; \pi\right]$ for the advantage estimator, which is approximated by optimizing

$$\arg\min_{\phi} \mathbb{E}_{(\boldsymbol{c},\boldsymbol{w})\sim p(\boldsymbol{c}),\pi_{\boldsymbol{\theta}_{\text{old}}}} \left[(V_{\phi}(\boldsymbol{c}) - R(\boldsymbol{w}, \boldsymbol{c}))^2 \right].$$

Unlike step-based methods, our approach leverages samples of the form $(\boldsymbol{c}, \boldsymbol{w}, R)$ that are generated per trajectory, where R is the trajectory performance.

3.2 Choice of the Deep Reinforcement Learning Algorithm

Training this type of policy can theoretically be done with most existing deep RL methods. Yet, learning in the parameter space requires to learn policies with a higher level of precision than in the step-based case. In the step-based case, smaller errors during action selection can still be corrected at a later time step. In ERL, we only select one parameter vector $\boldsymbol{w}$ per episode, hence no error correction of these parameters is possible. As a consequence, we chose TRPL since this method has shown to be more stable and precise than other RL methods as the trust regions for the policy update are implemented exactly. Furthermore, the trust regions are enforced per state while most other deep RL methods [15, 2, 25] only offer approximate trust region updates for which the trust region is enforced for the mean policy change over all states. We will now present the objective based on the adaptation of the TRPL algorithm to the ERL setup.

3.3 Movement Primitives as Parametrized Controllers

So far we only discussed how to learn the distribution in the parameter space, but not how to create actions. For this purpose, we use a trajectory generator $\Psi(\boldsymbol{w})$ that is typically translating the parameters $\boldsymbol{w}$ to a desired trajectory $\boldsymbol{\tau}^d = (\boldsymbol{s}_1^d, \boldsymbol{s}_2^d, \ldots, \boldsymbol{s}_T^d)$. Common choices of trajectory generators are DMPs [26, 27], *probabilistic movement primitives* (ProMPs) [28], or Viapoint MPs [29]. The step-based control signal is then retrieved by employing a trajectory tracking controller $f(\boldsymbol{s}_t, \boldsymbol{s}_t^d)$, such as a simple PD-controller that outputs torques given the current and desired position as well as velocity. Here, *no noise* is applied to the raw actions for exploration. Even more complex controllers such as Cartesian impedance controllers can be readily used.

As a result, trajectory generation and execution is independent of the policy training, enabling the use of any trajectory generator and tracking controller. For this work, we will use ProMPs [28] for trajectory generation and torque-based PD-controllers for action execution. In the case of a ProMP, we can optimize its weight vector that linearly influences the generated trajectories. Further, we can include additional parameters, such as the trajectory starting time as well as the execution speed of the motion (represented as velocity of the ProMP's phase variable) to our ERL optimization process. As our experiments show, this offers a powerful parametrization for tasks where exact timing is crucial, such as robot table tennis and throwing.

3.4 Advantages of Episode-Based Versus Step-Based Reinforcement Learning

Step-based RL is currently by far the more prominent perspective on RL. This seems a natural choice as step-based RL exploits the temporal structure of RL problems and is therefore expected to outperform ERL in terms of sample efficiency. Yet, step-based RL also comes with a few disadvantages. First, exploration in action space often results in a very jerky random walk behavior that does not

fully explore the trajectory space of the agent. In addition, the stochastic action selection also results in noisy returns, which translates into high variance of the policy gradient estimate. Second, the step-based exploration process also complicates the use of sparse or non-Markovian rewards in the step-based setting. While there do exist specialized solutions for some of these problems, they often require much more complex algorithms.

In contrast, ERL offers a simple framework for learning complex behavior with sparse and non-Markovian rewards, without special treatment of these cases. ERL only uses a fraction of data points to update the policy, since each trajectory is abstracted as only a single data point. While the small number of available training samples may initially appear to be a disadvantage, it offers three distinct benefits: (i) As the used controllers are deterministic, the returns used for the policy updates contain less variance than in the step-based case, which simplifies finding good policy updates. (ii) The exploration in parameter space allows for learning with sparse and non-Markovian rewards which can be used to define the desired behavior in a more direct way. Finally, (iii) the controller parametrization allows the inclusion of temporal parameters such as shifting or scaling of the desired trajectories. We believe that all these features constitute to superior performance of learned ERL policies in comparison to the step-based variant in most of our experiments. In conclusion, we consider ERL as a promising alternative to step-based RL in particularly for tasks where the desired behavior is hard to define with dense rewards.

4 Experimental Results

For our evaluation, we first demonstrate the benefits of deep ERL in terms of sparse rewards, precision, and energy efficiency. Afterwards, we conduct a large scale study on all 50 Meta-World tasks [30] to show that we indeed achieve improved precision on a variety of robot manipulation tasks. Lastly, we investigate multiple challenging control problems that are hard to solve in the step-based setting. We compare our method, which we call *BBRL-TRPL*, to step-based PPO [2], NDP [23] as well as to BBRL-PPO, which is equivalent to BBRL-TRPL but trained with PPO instead of TRPL.

Note that for NDP, the authors report the performance in terms of the used samples and not in terms of environment interactions (the NDP approach uses only every fifth interaction for training). We report the total number of environment interactions because we think this leads to a fairer comparison and also explains the rather poor performance of NDP in our experiments. For all tasks the context information $\boldsymbol{c}$ is always a subset of the observation space which is randomly initialized after each reset. The trajectory performance $R(\boldsymbol{w}, \boldsymbol{c})$ is, if not specified otherwise, the full undiscounted trajectory return. We always evaluate on 20 different seeds and compute 10 evaluation runs after each iteration. Following Agarwal et al. [31], we report the *interquartile mean* (IQM) with a 95% stratified bootstrap confidence interval as well as performance profiles where feasible (for hyperparameters see Appendix D).

4.1 Sparse Rewards Induce Energy Efficient Behavior

As introductory task we use an extension of the Mujoco reacher from OpenAI gym [32]. Instead of two actuated joints, we use five, but limit the context space, i. e. the location of the goal, to $y \geq 0$. This results in an increased control complexity with a slightly decreased task complexity. We investigate two types of rewards, a dense reward equivalent to the original reacher as well as a sparse reward that only provides the distance to the goal in the last time step of the episode. While BBRL-TRPL and BBRL-PPO can solve the task for both rewards, NDP cannot solve it for either of them (Figure 2 left and center left). PPO achieves a slightly better asymptotic performance than BBRL-TRPL in the dense setting, but is not able to consistently reach the goal for the sparse reward signal. However, after reviewing the learned policies in the dense setting, we found that PPO shows a better reward due to an limitation of the initial velocities in our ProMPs implementation which could be addressed by using different trajectory generators.

To demonstrate these capability for learning from sparse rewards in a more complex setting, we also evaluate the algorithms on a box pushing task. The goal is to move a box to a given goal location and

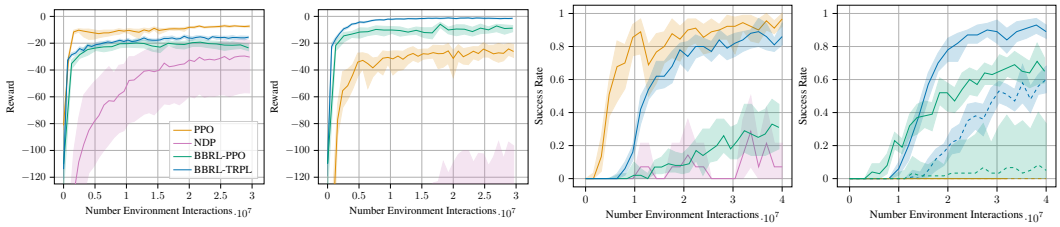

Figure 2: The learning curve for the modified reacher with dense (left) and sparse reward (center left) signal. The box pushing task is trained with three different rewards. The dense reward (center right) has the best performance for step-based PPO, while it performs poorly with the sparse reward in time (right, solid line) and the sparse reward in time and space (right, dashed line). BBRL-TRPL solves the task in the first two settings and even for second, most difficult sparse reward, it is still partially able to do so.

orientation using a simulated Franka Emika Panda (see Appendix B.1). We consider three rewards, a dense reward based on the goal and rotation distance for each time step and two sparse rewards. The first induces time-dependent sparsity by returning the reward signal only in the final time step. The second extends this to include space-dependent sparsity, and additionally requires the box to move near the goal in order to receive the reward. PPO can only solve this task in the dense setting (Figure 2 center right and right). NDP already struggles with the dense reward, thus, we did not evaluate it for the sparse rewards. BBRL-TRPL, on the other hand, learns the task in the dense and the first sparse setting without any problems. Even for the complex time and space sparsity, BBRL-TRPL is able to partially solve the task. BBRL-PPO receives merely moderate results for the dense and first sparse setting.

Yet, why would we use sparse rewards when dense rewards work well with step-based RL? A dense reward typically induces a rather fast motion with poor energy efficiency. As shown in our experiments, dense rewards do not work well if we want to generate accurate motions that reach the goal after a certain time and are energy efficient as well. Here, we can use sparse rewards instead, i. e., the distance to the goal is penalized only in the final time step, while the energy cost is accounted for in each time step.

To illustrate the trade-off between precision and energy efficiency, we analyzed the learned behaviors in both reward setups with different action penalty factors in the reward function. For each of these factors, we computed the average precision as well as the average energy consumption (Figure 3 left and center left). For all methods, decreasing the action penalty factor leads to a higher task precision. However, for the dense reward, a high task precision can only be achieved with high energy costs, while the sparse reward generates behavior of similar precision with one (box pushing) or even two (reacher) factors of magnitude less energy consumption. When analyzing the behaviors, it is visible that the dense reward behavior moves quickly to the target and stays there while the sparse reward behavior reaches the target only slightly before the specified end of the episode, resulting in a much slower, smoother and more energy efficient motion.

4.2 Large Scale Robot Manipulation

On the Meta-World benchmark suite [30], we showcase our ability to learn high quality policies. We train individual policies for each environment, but use the same hyperparameters. PPO achieves the best sample complexity, but converges to a slightly lower asymptotic performance than BBRL-TRPL (Figure 3 center right). While in the aggregated view the gap is rather small, the corresponding performance profiles (Figure 3 right) present a different view. Up to the 80% threshold, PPO and BBRL-TRPL perform similarly, however afterwards a gap starts to open up to a 15 percentage point lead for BBRL-TRPL. This means that BBRL-TRPL finds more consistent solutions with higher precision and solves these tasks without failures. Both, NDP and BBRL-PPO are not achieving a competitive performance.

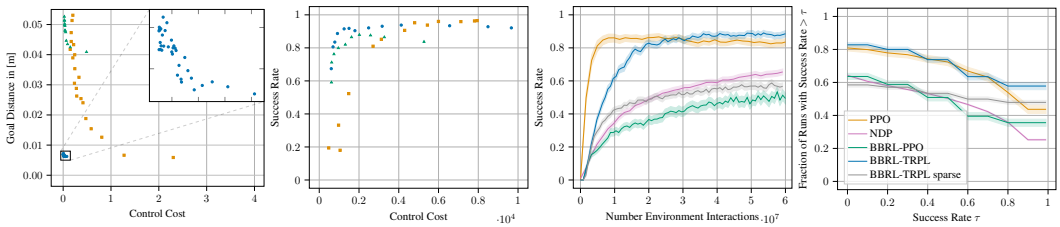

Figure 3: Energy efficiency (sum of squared control action costs) and task performance (distance to the target in the last time step or success rate) trade-off for reacher (left) and box pushing (center left). We average over 100 evaluation runs and all seeds and choose action penalty factors in the intervals $(0, 100]$ for the reacher and $(0, 1]$ for box pushing. For both tasks, BBRL-TRPL using sparse rewards can achieve a much higher energy efficiency (1 to 2 factors of magnitude) while achieving a similar target precision as PPO in the dense reward setting. The success rate (center right) for all 50 Meta-World tasks and the corresponding performance profile (right), i. e. the fraction of runs that perform above the threshold given by the x axis. While the sample efficiency is lower for BBRL-TRPL, the final quality of the policy is higher, which can also be seen at the fraction of runs that achieve 100% success rate (right).

As an additional ablation study, we trained BBRL-TRPL using a sparse reward (only the final reward of each episode is given), denoted as BBRL-sparse. While the IQM score is lower, we still complete 50% of the tasks with 100% success rate, which is higher than PPO using the dense reward. Moreover, the slope of the performance profile is rather small, i. e. almost all tasks that can be solved are solved to perfection. After further investigation, we found that poorly performing tasks all require behavior that involves sub-goals, e. g. bin picking, pick and place, etc. This could be addressed e. g. by sequencing several MPs and also giving the corresponding sub-goal as intermediate rewards, which is an interesting direction for future research.

4.3 Dealing with non-Markovian Rewards

As final evaluation, we demonstrate the effectiveness of our method in the presence of non-Markovian rewards. These rewards are particularly useful for complex robot learning tasks, where the whole trajectory history is needed to provide feedback to the agent (for environment details see Appendix B). We first use a modification of the hopper from Open AI gym [32], where we aim to jump as high as possible and land at a target location. The non-Markovian reward returns the maximum height and the minimum distance to the target during the episode. We compare our results to PPO trained with a Markovian reward which provides the height and goal distance in each time-step. Please note that we have done extensive reward shaping for PPO to gain the highest performance (i. e. maximum height and minimum goal distance). Overall, BBRL-TRPL achieves a higher jump height than PPO (Figure 4 left) accompanied with a lower goal landing distance (Figure 4 center left). BBRL-PPO can match the goal distance, but does not reliably learn the jump height. While BBRL-TRPL charges energy and then jumps just once, PPO tries to maximize the height in each time step which leads to several jumps during the episode (see Appendix C).

Another example where we benefit from non-Markovian rewards is a beer pong task [22] in which the agent must throw a ball into a cup at various locations on a table. The return is computed from the entire trajectory of the ball, e. g. by leveraging table contacts or the minimum distance to the cup. Since, we cannot directly train PPO on such a reward and designing a Markovian version is particularly difficult here, we fix the ball release time and consider the time between release of the ball and the end of the ball trajectory as final time step. This allows to compute the reward in a similar manner as for the non-Markovian setting. We evaluate BBRL-TRPL on the non-Markovian reward with a learned ball release time as additional controller parameter as well as PPO for the setting described above (Figure 3 right). BBRL-TRPL and BBRL-PPO both manage to throw the ball into the cup, while PPO struggles. Similar to the jumping task, BBRL-PPO exhibits a larger confidence interval as it is not able to consistently solve the task.

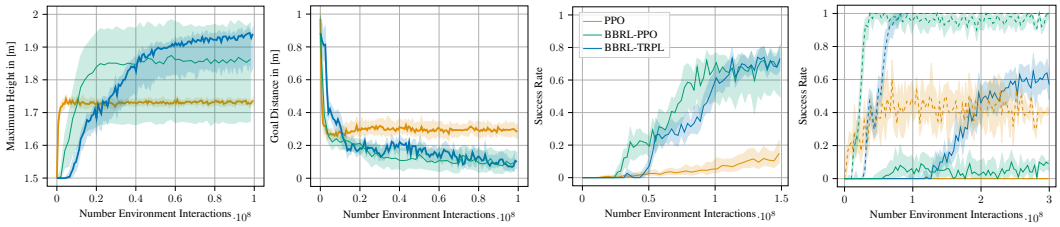

Figure 4: The maximum jumping height of the hopper's center of mass (left) and the goal landing position (center left). The non-Markovian reward allows to jump approximately 20 cm higher with increased goal precision. The beer pong task (center right) demonstrates a similar behavior. PPO has difficulties throwing the ball into the cup even when the optimal release point is fixed, whereas BBRL-TRPL can consistently succeed in the tasks with dynamic release points. The success rate of the table tennis task (right) for hitting the ball (dashed line) and successfully returning the ball within the vicinity of the target position (solid line). BBRL-TRPL consistently hits the ball and can also return it in the majority of cases. PPO is already struggling to hit the ball.

Lastly, we train an agent for simulated table tennis [22] where the context is four dimensional and given by features of the initial ball trajectory and the desired location for returning the ball. Intuitively, the agent should be rewarded if it hits the ball and returns the ball near the designated goal position. Similar to the throwing task, for PPO we consider the time after hitting the ball as one time step. For the BBRL approaches, we also learn the trajectory starting time as well as the speed of the desired trajectory (which is a parameter of the ProMP). Both parameters help to learn the precise timing required to play table tennis.

BBRL-TRPL always manages to hit the ball and successfully returns it within the vicinity of the goal in more than 60 % of the cases. While BBRL-PPO can hit the ball, it cannot return it with enough precision. We also tested BBRL-TRPL with fixed timing parameters (start time and velocity) and observed a similar behavior - the algorithm requires these timing parameters to learn how to put the ball at the desired location. PPO fails to even hit the ball consistently.

Summarizing, BBRL-TRPL is able to successfully deal with non-Markovian reward structures, which can be more intuitive, easier to define, and achieve better learned behaviors than engineered dense rewards.

5 Conclusion and Limitations

In this work, we proposed a new algorithm for ERL. Our method leverages the recently introduced TRPLs [9] to guarantee precise and robust policy updates in high-dimensional parameter spaces. In thorough empirical evaluations, we have shown that we can indeed learn policies with high precision even in the presence of sparse and non-Markovian rewards, where step-based approaches typically fail. Due to the use of sparse rewards, our learned behaviors are more energy efficient and less sensitive to high action costs, which leads to almost universally well-performing policies.

The main limitation of our method and ERL in general is that they typically require more interaction time than step-based RL in the dense reward setting. While we agree this can be critical in some scenarios, we still think the benefits outweigh this drawback in many scenarios. To address the sample complexity issue, we plan to investigate off-policy approaches of our method as well as re-planning of the motion primitive to obtain more training data for the policy update in future work. Another limitation of our method is that the desired trajectory is planned in advance in the beginning of the episode, and hence, cannot be altered during the execution. This might be problematic for unforeseen events or perturbations, i. e. highly complex or reactive behavior that cannot directly be modeled with the current motion representation. We believe that this limitation can also be tackled by incorporating re-planning the MP trajectory in the RL algorithm. For future work, we will also investigate sequencing multiple MPs to solve more complex tasks which are composed of sub-goals.

References

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

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. In *arXiv preprint*, 2017. URL http://arxiv.org/abs/1707.06347.

[3] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[4] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and trends in Robotics*, 2(1-2):388–403, 2013.

[5] A. Abdolmaleki, R. Lioutikov, J. R. Peters, N. Lau, L. Pualo Reis, and G. Neumann. Model-based relative entropy stochastic search. *Advances in Neural Information Processing Systems*, 28, 2015.

[6] C. Daniel, G. Neumann, and J. Peters. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pages 273–281. PMLR, 2012.

[7] A. Abdolmaleki, B. Price, N. Lau, L. P. Reis, and G. Neumann. Contextual covariance matrix adaptation evolutionary strategies. In *International Joint Conferences on Artificial Intelligence Organization)*, 2017.

[8] V. Tangkaratt, H. van Hoof, S. Parisi, G. Neumann, J. Peters, and M. Sugiyama. Policy search with high-dimensional context variables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[9] F. Otto, P. Becker, N. Anh Vien, H. C. Ziesche, and G. Neumann. Differentiable trust region layers for deep reinforcement learning. In *International Conference on Learning Representations*, 2021.

[10] F. Sehnke, C. Osendorfer, T. Rückstiess, A. Graves, J. Peters, and J. Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 21(4):551–559, May 2010. doi:10.1016/j.neunet.2009.12.004.

[11] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

[12] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[13] S. Mannor, R. Y. Rubinstein, and Y. Gat. The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 512–519, 2003.

[14] A. Abdolmaleki, D. Simões, N. Lau, L. P. Reis, and G. Neumann. Contextual direct policy search. *Journal of Intelligent & Robotic Systems*, 96(2):141–157, 2019.

[15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust Region Policy Optimization. In *Proceedings of Machine Learning Research*, pages 1889–1897, 2015. URL http://proceedings.mlr.press/v37/schulman15.html.

[16] H. F. Song, A. Abdolmaleki, J. T. Springenberg, A. Clark, H. Soyer, J. W. Rae, S. Noury, A. Ahuja, S. Liu, D. Tirumala, N. Heess, D. Belov, M. Riedmiller, and M. M. Botvinick. V-mpo: On-policy maximum a posteriori policy optimization for discrete and continuous control. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylOlp4FvH.

[17] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO. In *International Conference on Learning Representations*, 2020. URL http://arxiv.org/abs/2005.12729.

[18] J. Kober and J. Peters. Policy search for motor primitives in robotics. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2008/file/7647966b7343c29048673252e490f736-Paper.pdf.

[19] F. Stulp and O. Sigaud. Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1547–1554, Madison, WI, USA, 2012. Omnipress. ISBN 9781450312851.

[20] F. Stulp and O. Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. 2012.

[21] A. Kupcsik, M. P. Deisenroth, J. Peters, L. A. Poha, P. Vadakkepata, and G. Neumann. Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439, 2017. doi:10.1016/j.artint.2014.11.005. URL http://eprints.lincoln.ac.uk/25774/1/Kupcsik_AIJ_2015.pdf. Impact Factor: 3.333.

[22] O. Celik, D. Zhou, G. Li, P. Becker, and G. Neumann. Specializing versatile skill libraries using local mixture of experts. In *Conference on Robot Learning*, pages 1423–1433. PMLR, 2022.

[23] S. Bahl, M. Mukadam, A. Gupta, and D. Pathak. Neural dynamic policies for end-to-end sensorimotor learning. *Advances in Neural Information Processing Systems*, 12 2020. URL https://arxiv.org/abs/2012.02788v1.

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

[25] R. Akrour, J. Pajarinen, J. Peters, and G. Neumann. Projections for approximate policy iteration algorithms. In *Proceedings of Machine Learning Research*, pages 181–190, 2019.

[26] S. Schaal. *Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics*, pages 261–280. Springer Tokyo, Tokyo, 2006. ISBN 978-4-431-31381-6. doi:10.1007/4-431-31381-8_23. URL https://doi.org/10.1007/4-431-31381-8_23.

[27] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors. *Neural Computation*, 25(2), 2013.

[28] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. Probabilistic movement primitives. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/e53a0a2978c28872a4505bdb51db06dc-Paper.pdf.

[29] Y. Zhou, J. Gao, and T. Asfour. Learning via-point movement primitives with inter-and extrapolation capabilities. In *International Conference on Intelligent Robots and Systems)*, pages 4301–4308. IEEE, 2019.

[30] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

[31] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

[32] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.