Black box meta-learning intrinsic rewards for sparse-reward environments

Octavio Pappalardo[1], Juan M. Santos[2], and Rodrigo Ramele[3]

[1]Universidad de Buenos Aires , Facultad de Ciencias Exactas y Naturales, Departamento de Física ,
octaviopappalardo@gmail.com
[2]Universidad Nacional de Hurlingham, Centro de Investigacion en Informatica Aplicada ,
juan.santos@unahur.edu.ar
[3]Instituto Tecnológico de Buenos Aires, Departamento de Ingeniería Informática ,
rramele@itba.edu.ar

Abstract: Despite the successes and progress of deep reinforcement learning over the last decade, several challenges remain that hinder its broader application. Some fundamental aspects to improve include data efficiency, generalization capability, and ability to learn in sparse-reward environments, which often require human-designed dense rewards. Meta-learning has emerged as a promising approach to address these issues by optimizing components of the learning algorithm to meet desired characteristics. Additionally, a different line of work has extensively studied the use of intrinsic rewards to enhance the exploration capabilities of algorithms. This work investigates how meta-learning can improve the training signal received by RL agents. The focus is on meta-learning intrinsic rewards under a framework that doesn't rely on the use of meta-gradients. We analyze and compare this approach to the use of extrinsic rewards and a meta-learned advantage function. The developed algorithms are evaluated on distributions of continuous control tasks with both parametric and non-parametric variations, and with only sparse rewards accessible for the evaluation tasks.

1 Introduction

The adoption of neural networks and recent algorithmic advances in reinforcement learning (RL) have enabled its application to complex decision-making problems [1, 2, 3, 4]. However, several challenges persist that must be addressed before RL can be effectively applied to broader range of domains, especially those requiring interaction with the real world.

Several sub-fields of research have emerged out of this need [5, 6, 7, 8, 9, 10, 11]. Two main issues they target are poor data efficiency in task learning and the limited generalization capabilities of learned policies when applied to new tasks. Related to data efficiency, another central challenge in RL is devising efficient exploration strategies, and achieving an appropriate trade-off with exploitation.

Reinforcement Learning: In this work, we deal with discrete-time finite-horizon discounted Markov decision processes (MDPs). Reinforcement learning agents learn a policy π_θ, which maps states to a probability distribution over actions, through interaction with an environment to maximize expected cumulative reward. The interaction takes place in episodes of length T. The state of the environment at step t is $s_t \in S$, the action the agent takes is $a_t \in A$ and the reward it receives is r_t. The trajectory of an agent throughout an episode is denoted by $\tau = (s_0, a_0, r_1, \ldots, s_T)$ and the objective function it seeks to maximize is $J(\pi) = \mathbb{E}_{\tau\sim p(\tau)}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t\right] = \mathbb{E}_{\tau\sim p(\tau)}[G(\tau)]$, where γ is the MDP's discount factor and $G(\tau)$ is the discounted cumulative return attained in an episode of trajectory τ.

Meta Reinforcement Learning: An agent's learning algorithm can be viewed as a mapping from the data generated by interacting in an environment, $\mathcal{D}$, to a policy: $\theta = f(\mathcal{D})$, where θ represents the parameters of a parameterized policy π_θ. The learning algorithm f itself depends on a variety of decisions. If we parameterize the choice of a subset of this decisions with parameters ϕ, then we can make the dependence explicit, $\theta = f(\mathcal{D}; \phi) = f_\phi(\mathcal{D})$. Meta-RL methods learn ϕ, part of the learning algorithm, such that it becomes more effective when applied to new tasks [12]. A common setting for these methods involves a distribution of tasks $p(\mathcal{M})$ and an objective that drives meta-learning to maximize the expected cumulative return an agent attains throughout its lifetime when interacting with sampled tasks $\mathcal{M}^i \sim p(\mathcal{M})$ while learning with f_ϕ. We define an agents *lifetime* as its whole interaction with an environment (which spans multiple episodes). If the return attained at each episode of interaction with the environment is equally weighted by the objective, then it is described by equation 1, but variations with different weightings are possible [11].

$$\mathcal{J}(\phi) = \mathbb{E}_{\mathcal{M}^i \sim p(\mathcal{M})} \left[\mathbb{E}_{\mathcal{D}} \left[\sum_{\tau \in \mathcal{D}} G(\tau) \;\middle|\; f_\phi, \mathcal{M}^i \right] \right]. \tag{1}$$

The meta-learning process operates at two different levels: In the inner loop, given a task $\mathcal{M}^i$, the learning algorithm f_ϕ learns a task-specific policy π_θ through interaction with the task. In the outer loop a meta-learning algorithm learns parameters ϕ of f using data from multiple inner loops.

Intrinsic rewards: For many RL algorithms, directly maximizing extrinsic rewards can lead to poor exploration of the environment; specially when dealing with sparse-reward environments. The use of intrinsic reward signals, r^i, (that either complement or replace extrinsic rewards $r = r^e$) has occupied a central role in searching for ways to achieve better exploration and balance it with exploitation [13].

In this work we combine these two approaches by meta-learning an intrinsic reward function. For this purpose, we model the reward function itself as a stochastic agent whose 'actions' are the rewards it gives at each step and train it with a standard RL algorithm. Our experiments evaluate whether this approach is beneficial compared to learning with extrinsic rewards. Additionally, we contrast it against the use of meta-learned advantage function in the training signal. All experiments are conducted on distributions of continuous control tasks where there is access to shaped extrinsic rewards during meta-learning, but only sparse rewards in the test environments.

2 Related Work

One popular way to optimize for fast task learning is to use a neural network that uses all the data collected up to step t to determine its action distribution at that step. During meta-training, the network is trained to leverage this interaction history to select appropriate actions for the current task. This family of methods was first introduced in [14] and [15].

Instead of meta-training a network to directly output the action taken at each step, another family of methods uses an RL algorithm in the inner loop and learns some component of this inner loop procedure that impacts the final performance. Most work restrict themselves to components that have a differential influence over the policy's parameters and use meta-gradients for meta-training [16]. This framework was introduced in MAML [17], which meta-learns the initial parameters of the policy. Many other components have also been explored [18, 19, 20, 21]. Some methods avoid the use of meta-gradients: [22, 17] consider first order approximations of meta-gradients , [23] uses a value based method, and [24] uses an evolutionary algorithm.

Several methods exist for designing intrinsic rewards [13], most of which are heuristic-based. Among these, some approaches aim to reduce the agent's uncertainty about specific quantities, while others encourage visiting new or varied states. However, these are not the only possibilities. This work studies a different approach, aiming to *learn* the intrinsic motivation signal [25].

Previous work has learned an intrinsic reward signal using meta-gradients: [26, 27] do so in the context of a single task, while [28] does this with a distribution of tasks. Additionally, [29] explores meta-learning a reward function with a discrete search over a space of programs. Rather than replacing extrinsic rewards, [30] learns to shape them into a denser signal. Among these, [28] shares the closest algorithmic similarity to our work. The main differences being that they consider a deterministic reward function, train it using meta-gradients, and conduct all experiments in grid world environments.

Beyond the aforementioned work, several other methods have studied meta-learning different parameterizations of the training signal beyond intrinsic rewards [31, 32, 33, 34, 35].

3 Black Box Meta-learning Intrinsic Reward Signals

This work explores the meta-training of learning signals, which are used to train policies by quantifying their fitness. We consider the inner loop learning algorithm f to be a reinforcement learning algorithm. Instead of training with the standard RL objective, in our approach f uses an objective that is partly determined by a meta-learned neural network. This meta-learned network is itself also trained using RL to maximize the objective defined in equation 1. Unlike previous methods that rely on meta-gradients to do so, our approach avoids the computation of second-order gradients. We achieve this by not modelling explicitly the structure of the inner loop learning; instead we let the outer loop treats it as part of the problem's stochasticity. As a result, this approach doesn't require the ability to compute gradients of the policy's parameters with respect to the meta-learned parameters. Due to this considerations we refer to the method as being *black box*.

We now describe in more detail how we meta-train the intrinsic reward function. The intrinsic reward function is modelled as a stochastic agent $\pi^r_\phi\left(r^i_t \mid \mathcal{D}_{:t}\right)$, where $\mathcal{D}_{:t}$ encodes all the interaction in the MDP up to time-step t. We use a LSTM [36] for this purpose. At each step t, the LSTM receives as input the tuple $\{s_t, a_t, \pi_\theta(a_t \mid s_t), r^e_t, r^i_{t-1}, \pi^r_\phi(r^i_{t-1} \mid \mathcal{D}_{:t-1}), d_t\}$ where: π_θ is the policy that is being trained, r^e_t is the extrinsic reward received from the environment at time-step t , $d_t \in \{0, 1\}$ indicates whether t is the start of a new episode, r^i_{t-1} is the intrinsic reward delivered by π_ϕ at time-step $t-1$ and $\pi^r_\phi(r^i_{t-1} \mid \mathcal{D}_{:t-1})$ is its probability density. We train this network with PPO [37] to maximize the objective defined in equation 1, modified to add per episode discounting (A.2.2). Because the network's output influences the inner loop performance, during meta-training it learns to use the history of interaction with the environment to generate outputs that suit the current task. The resulting optimization scheme is similar to that of RL^2 [14, 15] but applied to a different space. The data with which it trains comes from running multiple inner loops. For a simplified, high-level overview of the proposed method, refer to the pseudocode in Algorithm 1; further implementation details are discussed in later sections. We apply the same framework in section 5.2 when learning an advantage function.

As mentioned, a salient feature of our approach compared to previous methods is that it avoids the need to compute second-order gradients by bypassing the explicit modeling of the effect the meta-learned signal has over the policy's parameters. We now discuss some advantages and disadvantages of this black box approach versus the use of meta-gradients.

The main potential drawback is that, by explicitly considering the role of intrinsic rewards in the inner loop, meta-gradients can provide a less noisy signal for meta-learning. This is discussed in [38] in the context of meta-learning policy parameters. By comparison, the proposed method is simpler because it is framed as a standard RL problem and doesn't require the calculation of gradients through an optimization process. Moreover, calculating second-order gradients is a computationally expensive operation compared to first-order gradients, making the black box approach favorable in this regard. Another related advantage is that the method is indifferent to how the inner loop uses the meta-learned component. In particular, it can be used in a non-differentiable manner to affect the choice of actions; whereas meta-gradient methods cannot be applied in such situations.

Algorithm 1 Meta-learning Intrinsic Rewards

```
 1: Initialize intrinsic reward agent $\pi^r_\phi$, outer loop critic, and task distribution $p(\mathcal{M})$
 2: while $\phi$ not converged do
 3:     $\mathcal{D}_{\text{outer loop}} \leftarrow \emptyset$
 4:     batch_of_tasks $\leftarrow$ sample tasks from $p(\mathcal{M})$
 5:     for each task $\mathcal{M}^i$ in batch_of_tasks do
 6:         Initialize policy $\pi_\theta$ and inner loop critic
 7:         $\mathcal{D}_{\text{life}} \leftarrow \emptyset$
 8:         while lifetime not finished do
 9:             $\mathcal{D}_{\text{inner loop}} \leftarrow$ collect data in $\mathcal{M}^i$ using $\pi_\theta$
10:             Update $\pi_\theta$ and inner loop critic with $\mathcal{D}_{\text{inner loop}}$ as in PPO, replacing environment rewards
                with intrinsic rewards from $\pi^r_\phi$
11:             $\mathcal{D}_{\text{life}} \leftarrow \mathcal{D}_{\text{life}} \cup \mathcal{D}_{\text{inner loop}}$
12:         end while
13:         $\mathcal{D}_{\text{outer loop}} \leftarrow \mathcal{D}_{\text{outer loop}} \cup \mathcal{D}_{\text{life}}$
14:     end for
15:     Update $\pi^r_\phi$ and outer loop critic with $\mathcal{D}_{\text{outer loop}}$ as in PPO with respect to the objective in
        Equation 1
16: end while
```

4 Experimental Setup

4.1 Benchmarks

We conduct all our experiments using the MetaWorld benchmarks [39]. These consists of distributions of continuous control tasks where a simulated robotic arm must interact with an object to achieve a desired configuration. Variations among tasks can be either non parametric (changes in the class of problem, e.g., opening a window, closing a drawer) or parametric (changes in goal positions or initial positions within the same class of problem):

- ML1 benchmarks operate within a single class and contain 50 randomly sampled parametric variations for training and another 50 for evaluation.
- ML10 benchmark consists of a set of 10 classes of problems for training and 5 different ones for evaluation. For each class, 50 sampled parametric variations are considered.

All tasks share the same 39-dimensional observation space and 4-dimensional action space. Crucially, the agent lacks knowledge of both the task class and the goal position; it must infer these from interaction. All episodes are 500 steps long without the possibility of early termination (even if the task is completed). Further details of the utilized benchmarks can be found in the supplementary material A.1.

Availability of Rewards: It is often easier and more practical to define desired behaviour through sparse rewards (e.g. positive signal upon completion of an objective). However, the absence of frequent rewards makes learning significantly more challenging. This often leads to the manual design of reward signal, which can be a labor-intensive process and increase the risk of reward hacking [40]. This work considers a hybrid setting in which there is access to shaped rewards for the training tasks but only to sparse rewards during evaluation. In practice, this means we use shaped rewards during meta-training to compute the objective in equation 1, but exclusively the sparse rewards as inputs to the meta-learned recurrent network. Some prior work that operates in this same setting includes [19, 41, 42]. The sparse rewards used are: -0.2 in the last step of episodes where the agent failed, and $1 - 0.7\frac{\text{num. executed steps}}{\text{T}}$ on steps where the agent reaches the goal configuration (and has not done so previously within the episode).

4.2 Implementation Details

Two different neural network architectures were used in this work: an MLP for the policies and inner loop critics, and an LSTM for the meta-trained networks that output training signals and act as outer loop critics. A learning rate of 3×10^{-4} was used in the inner loop and 5×10^{-5} in the outer loop. We used the Adam optimizer, tanh activations, and orthogonal initialization. The outputs of the policies and the training signal generators are modeled as Gaussian distributions, with the policies assuming zero covariance. During training, each lifetime used only 4000 steps of data for policy learning. Each outer loop update used data from 30 inner loops.

Further implementation details can be found in the supplementary material A.2 and in the project's code[1].

4.3 Evaluation Methodology

During evaluation, the intrinsic reward function was used deterministically by considering the mean of the output Gaussian. In some experiments, making the policy deterministic after in-task training was also found to be beneficial.

The metric used for evaluation is the percentage of episodes in which the agent succeeded (reached a goal configuration).

The values reported in the results section for each algorithm are the average performance obtained from different instances of the method (each trained from scratch with a different seed), along with the standard deviation. For each benchmark, a given seed determines the set of parametric variations used. Five seeds were used for methods that utilize extrinsic rewards, and three for the meta-learning methods; larger numbers were not used due to the high computational demands of meta-RL algorithms and compute constraints.

Furthermore, for each seed, the method's performance on a given set of tasks (train or test set) was obtained as the average performance over multiple evaluation runs. This was done by running the method on each task of the set 10 independent times and averaging the performance across all these runs. For example, for the ML10 evaluation set, which has 5 different problem classes, each with 50 parametric variations, a total of 2500 runs were considered to obtain the performance of each seed of a method.

5 Experimental results

This section presents the main experimental results, divided into two parts. The first part compares training with meta-learned intrinsic rewards to training with extrinsic rewards. The second part investigates the effect of learning a different parameterization of the training signal.

5.1 Intrinsic vs Extrinsic Rewards

This set of experiments evaluates whether there is benefit to be gained in making an RL agent train with learned intrinsic rewards. To this end, we compared the success rate of a PPO agent when trained with shaped extrinsic rewards, sparse extrinsic rewards, and intrinsic rewards generated by a meta-learned network (which only has access to the environment's sparse rewards).

Evaluations were conducted over the ML1-reach, ML1-close-door, and ML1-button-press benchmarks, allowing only 4000 steps of learning. Figures 1 and 2 illustrate the results. Both figures demonstrate substantial improvements when training with an intrinsic reward function. Figure 1 evidences the choice of using an RL algorithm that trains with batches of 4 episodes of data (A.2.1). It also reflects that making the policy deterministic after training can improve performance in some environments. Figure 2 shows that there is no decline in performance when moving from the tasks

[1] Our implementation is available at: https://github.com/Octavio-Pappalardo/Meta-learning-rewards

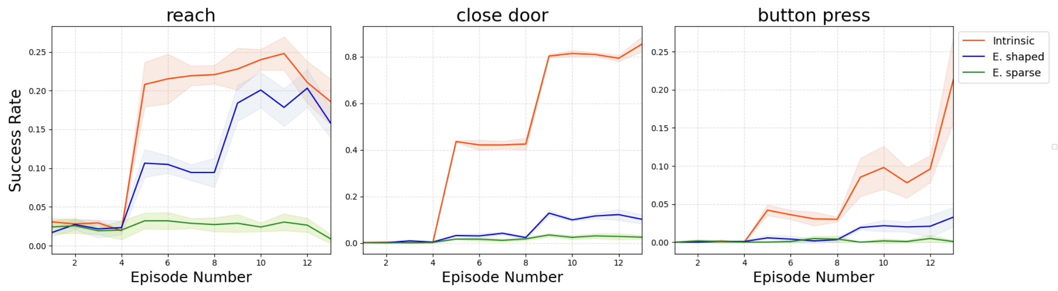

Figure 1: Comparison of the average performance of agents as they interact with tasks from the test set. The values and their standard deviations (represented by the shaded region) were obtained as explained in section 4.3. The success rate when training agents using three different types of rewards is shown: intrinsic (red), shaped extrinsic (blue), and sparse extrinsic(green). Three benchmarks are considered: ML1-reach, ML1-close-door, and ML1-button-press. The last episode reflects the performance of the final policy when made deterministic.

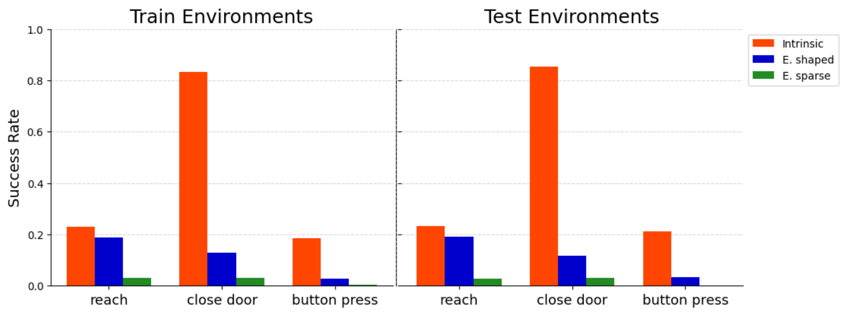

Figure 2: Success rate of agents trained with different rewards on various meta-learning benchmarks, ML1-reach, ML1-close-door, and ML1-button-press, after an adaptation period of 4000 steps. The figure compares the performance when using three different types of rewards: intrinsic (red), shaped extrinsic (blue), and sparse extrinsic(green). The values were obtained as explained in the section 4.3.

in the training set to the test set. While this is expected when training with extrinsic rewards (since no meta-learned component is applied), the fact that the same behavior occurs when training with intrinsic rewards indicates that the learned reward network effectively generalizes to new, unseen environments.

However, the observed benefits do not come without costs. The method using intrinsic rewards required having a prior meta-learning phase, which can be computationally costly and is contingent upon having access to training tasks that share structure with those in the test set.

As mentioned, using intrinsic rewards significantly improves performance compared to both types of extrinsic reward signals. It is worth noting that the fairer comparison in the considered evaluation setting, where the meta-learned network does not have access to shaped rewards, is against training with the sparse extrinsic rewards (which showed little to no progress). This makes the gains more notable. Finally, outperforming the shaped extrinsic rewards suggests that learning rewards can also be used for the design or improvement of the shaped reward signals that standard RL benchmarks typically use [30].

5.2 Intrinsic Rewards vs Learned Advantages

In this section, we consider whether intrinsic rewards are the right component to meta-learn and we discuss some other options. Focusing on intrinsic rewards is appealing for at least two reasons: 1) as mentioned in section 2, the use of intrinsic rewards is a well-studied topic in RL and has consistently shown benefits, 2) all standard RL algorithms assume they receive rewards; thus, meta-learned rewards can be directly integrated into any of them while maintaining the algorithm's structure.

As discussed in section 2, it is possible to meta-learn several other parameterizations of the RL objective. In this section, we explore another such parameterization by meta-learning an advantage function. Instead of learning to assign partial credit to a transition and its preceding transitions, the network learns to evaluate each transition's goodness independently. This set of experiments also introduces non-parametric variations among environments. Results are shown in figure 3.

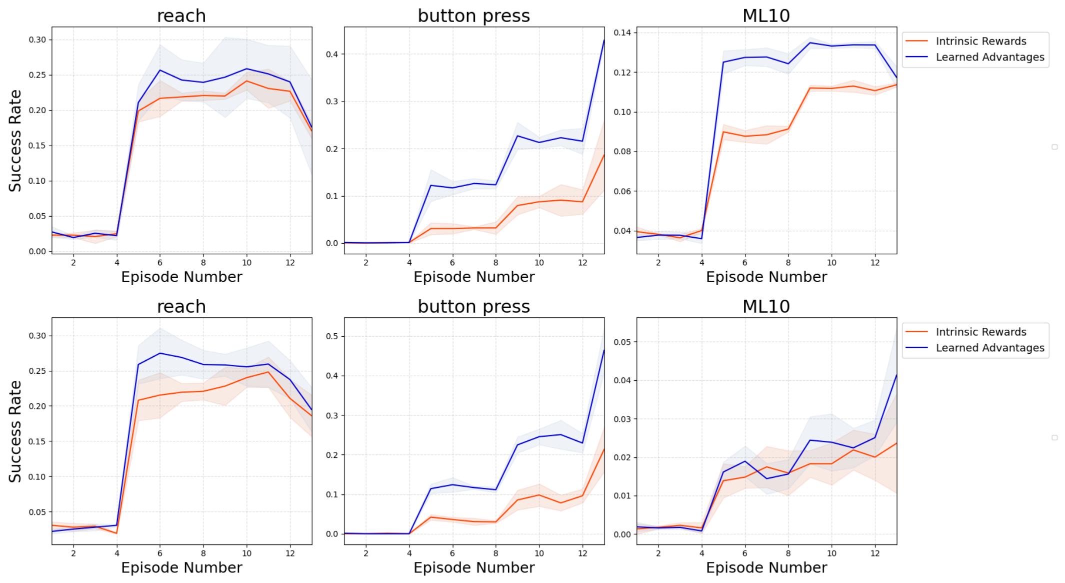

Figure 3: Comparison of the average performance of meta-learning methods as they interact with a new task from the training set (top row) and the test set (bottom row). Success rates are shown for two methods that meta-learned different parameterizations of the loss: intrinsic rewards (red) and advantages (blue). Three benchmarks are considered: ML1-reach, ML1-button-press, and ML10. The last episode reflects the performance of the final policy when made deterministic.

Both methods exhibit similar qualitative behaviour. For the benchmarks considered, using the learned advantage function shows some benefits, which are statistically significant only for ML1-button-press and for the training tasks of ML10. While both methods show good generalization when facing parametric variations, they achieve a very low success rate on the test tasks of ML10 (although some performance improvement occurs with updates).

Meta-learning to predict the training signal can be complementary to learning other components of the inner loop. Training signals hold knowledge on the goodness of observed behaviours [28]. Other components can address different loci of knowledge and provide additional benefits. In particular, the most popular approach in meta-learning literature is to meta-learn parameters of a policy. In the supplementary material we show the performance of two such methods when applied to our experimental setting. One advantage of meta-learning part of the objective function is its applicability to broad distributions of environments and agents. It has also shown to be adequate for the many-shot meta RL setting [32, 31]. Determining the best combination of components to meta-learn remains an open problem and likely to depend on the specific setting.

6 Conclusions

This paper presented a meta-learned stochastic intrinsic reward function modeled as a recurrent network and trained analogously to a standard reinforcement learning agent. The network was trained through interactions with a distribution of training tasks and evaluated in new, sparse-reward environments. Our experiments demonstrated that a policy that trains using these intrinsic rewards significantly accelerate it's learning compared to one using extrinsic rewards, whether shaped or sparse. The learned reward signal demonstrated effective generalization across parametric task variations, though this was not the case when confronting generalization among a small set of distinct problem types. Additionally, we investigated alternatively learning a policy using a meta-learned advantage function, achieving slightly better results. Throughout the paper, we discussed the benefits and drawbacks of different features of our approach.

Potential future directions that stem from this work include: conducting a quantitative analysis comparing our black box approach with the use of meta-gradients for the outer loop, extending the method to longer lifetimes and broader task distributions, combining or comparing with the meta-learning of other components, and applying it in settings where only sparse rewards are available during meta-training. Some straightforward avenues for improvement to our approach include using a network that has access to future interaction steps within the batch of collected data for generating rewards, and computing rewards for off-policy data to enable data reuse.

References

[1] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[2] D. J. Mankowitz, A. Michi, A. Zhernov, M. Gelmi, M. Selvi, C. Paduraru, E. Leurent, S. Iqbal, J.-B. Lespiau, A. Ahern, et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

[3] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[4] J. Luo, C. Paduraru, O. Voicu, Y. Chervonyi, S. Munns, J. Li, C. Qian, P. Dutta, J. Q. Davis, N. Wu, X. Yang, C.-M. Chang, T. Li, R. Rose, M. Fan, H. Nakhost, T. Liu, B. Kirkman, F. Altamura, L. Cline, P. Tonker, J. Gouker, D. Uden, W. B. Bryan, J. Law, D. Fatiha, N. Satra, J. Rothenberg, M. Waraich, M. Carlin, S. Tallapaka, S. Witherspoon, D. Parish, P. Dolan, C. Zhao, and D. J. Mankowitz. Controlling commercial cooling systems using reinforcement learning, 2022.

[5] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. Model-based reinforcement learning: A survey, 2022.

[6] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

[7] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[8] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function, 2018.

[9] M. Laskin, A. Srinivas, and P. Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5639–5650. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/laskin20a.html`.

[10] N. Vithayathil Varghese and Q. H. Mahmoud. A survey of multi-task deep reinforcement learning. *Electronics*, 9(9), 2020. ISSN 2079-9292. doi:10.3390/electronics9091363. URL `https://www.mdpi.com/2079-9292/9/9/1363`.

[11] J. Beck, R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

[12] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[13] A. Aubret, L. Matignon, and S. Hassas. A survey on intrinsic motivation in reinforcement learning, 2019.

[14] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[15] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[16] Z. Xu, H. van Hasselt, and D. Silver. Meta-gradient reinforcement learning, 2018.

[17] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[18] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint arXiv:1909.09157*, 2019.

[19] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. Meta-reinforcement learning of structured exploration strategies. *Advances in neural information processing systems*, 31, 2018.

[20] Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[21] E. Park and J. B. Oliva. Meta-curvature. *Advances in neural information processing systems*, 32, 2019.

[22] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms, 2018.

[23] F. Sung, L. Zhang, T. Xiang, T. Hospedales, and Y. Yang. Learning to learn: Meta-critic networks for sample efficient learning, 2017.

[24] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. Jonathan Ho, and P. Abbeel. Evolved policy gradients. *Advances in Neural Information Processing Systems*, 31, 2018.

[25] S. Singh, R. Lewis, and A. Barto. Where do rewards come from? 01 2009.

[26] Z. Zheng, J. Oh, and S. Singh. On learning intrinsic rewards for policy gradient methods. *Advances in Neural Information Processing Systems*, 31, 2018.

[27] B. Stadie, L. Zhang, and J. Ba. Learning intrinsic rewards as a bi-level optimization problem. In J. Peters and D. Sontag, editors, *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 111–120. PMLR, 03–06 Aug 2020.

[28] Z. Zheng, J. Oh, M. Hessel, Z. Xu, M. Kroiss, H. Van Hasselt, D. Silver, and S. Singh. What can learned intrinsic rewards capture? In *International Conference on Machine Learning*, pages 11436–11446. PMLR, 2020.

[29] F. Alet, M. F. Schneider, T. Lozano-Perez, and L. P. Kaelbling. Meta-learning curiosity algorithms. *arXiv preprint arXiv:2003.05325*, 2020.

[30] H. Zou, T. Ren, D. Yan, H. Su, and J. Zhu. Reward shaping via meta-learning, 2019.

[31] L. Kirsch, S. van Steenkiste, and J. Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

[32] J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. van Hasselt, S. Singh, and D. Silver. Discovering reinforcement learning algorithms, 2021.

[33] W. Zhou, Y. Li, Y. Yang, H. Wang, and T. Hospedales. Online meta-critic learning for off-policy actor-critic methods. *Advances in neural information processing systems*, 33:17662–17673, 2020.

[34] Z. Xu, H. P. van Hasselt, M. Hessel, J. Oh, S. Singh, and D. Silver. Meta-gradient reinforcement learning with an objective discovered online. *Advances in Neural Information Processing Systems*, 33:15254–15264, 2020.

[35] S. Bechtle, A. Molchanov, Y. Chebotar, E. Grefenstette, L. Righetti, G. Sukhatme, and F. Meier. Meta learning via learned loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, 2021.

[36] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[38] B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever. Some considerations on learning to explore via meta-reinforcement learning, 2019.

[39] T. Yu, D. Quillen, Z. He, R. Julian, A. Narayan, H. Shively, A. Bellathur, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021.

[40] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety, 2016.

[41] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019.

[42] T. Z. Zhao, A. Nagabandi, K. Rakelly, C. Finn, and S. Levine. Meld: Meta-reinforcement learning from images via latent state models. *arXiv preprint arXiv:2010.13957*, 2020.

A Appendix Experimental Setup

Section A.1 gives more information on the Meta World benchmarks involved in the study. Section A.2 sections provide more detail on the algorithms implementations.

A.1 Benchmarks

This section gives more information on the utilized benchmarks [39]. They all are from version 2 of MetaWorld.

Observation and action space: The 4-dimensional action space contains 3 dimensions for the change in 3D space of the robotic arm's end-effector, plus 1 dimension for the gripper's normalized torque. All dimensions are bounded to the range $[-1, 1]$. The 39-dimensional observation space consists of the 3D Cartesian position of the end-effector, a normalized measurement of how open the gripper is, the 3D position and quaternion of the first and second object of interest, all this same information from the previous time step, and finally the 3D position of the goal. In the benchmarks we utilized, the 3D position of the goal is zeroed out which forces the agent to learn to recognize and adapt to the task through interaction. For tasks where there isn't a second object of interest this information is also zeroed out.

ML1 benchmarks: The choice of these benchmarks over others in the ML1 category was arbitrary except for the condition that the agent trained with shaped extrinsic rewards exhibited some progress during its lifetime.

- **'ML1 reach'** Reach a goal position. Randomize the goal positions.
- **'ML1 close door'** Close a door with a revolving joint. Randomize door positions.
- **'ML1 button press'** Press a button. Randomize button positions

ML10 benchmark:

Training environments

- insert peg side: Insert a peg sideways. Randomize peg and goal positions.
- reach-v2: Reach a goal position. Randomize the goal positions.
- door-open-v2: Open a door with a revolving joint. Randomize door positions.
- basketball-v2: Dunk the basketball into the basket. Randomize basketball and basket positions.
- open window: Push and open a window. Randomize window positions.
- pick & place: Pick and place a puck to a goal. Randomize puck and goal positions.
- button-press-topdown-v2: Press a button from the top. Randomize button positions.
- push-v2: Push the puck to a goal. Randomize puck and goal positions.
- close drawer: Push and close a drawer. Randomize the drawer positions.
- sweep-v2: Sweep a puck off the table. Randomize puck positions.

Test environments

- door-close-v2: Close a door with a revolving joint. Randomize door positions.
- place onto shelf: Pick and place a puck onto a shelf. Randomize puck and shelf positions.
- drawer-open-v2: Open a drawer. Randomize drawer positions.
- sweep into hole: Sweep a puck into a hole. Randomize puck positions.
- pull lever: Pull a lever down 90 degrees. Randomize lever positions.

A.2 Implementation Details

In section A.2.1 tables with the values of different hyperparameters for each method are presented. Their nomenclature matches that used in the project's code for easier reproducibility. Then, in section A.2.2 we give a discussion on the influence of various hyperparameters, and provide some additional information.

A.2.1 Hyperparameter Tables

PPO agent/ Inner loop

Hyperparameters for the inner loop PPO agent. The hyperparameters are the same for all variations of training signals: extrinsic shaped rewards, extrinsic sparse rewards, intrinsic learned rewards, and learned advantage function.

Hyperparameter	**Value**
`total_timesteps`	`6000`
`num_steps`	`2000`
`learning_rate`	`3e-4`
`adam_eps`	`1e-5`
`gamma`	`0.99`
`gae`	`True`
`gae_lambda`	`0.95`
`ppo.update_epochs`	`64`
`ppo.num_minibatches`	`16`
`ppo.normalize_advantage`	`True`
`ppo.clip_coef`	`0.2`
`ppo.entropy_coef`	`0.0`
`ppo.valuef_coef`	`0.5`
`ppo.clip_grad_norm`	`True`
`ppo.max_grad_norm`	`0.5`
`ppo.target_KL`	`None`

Table 1: Table with the hyperparameter configuration used for task adaptation. Valid both for when the PPO agent was runned on its own. as well as when it used meta-learned training signals. hyperparameters were also the same during meta-training phases.

Learned Intrinsic rewards and Advantage function/ Outer loop

Hyperparameter	**Value**
`num_epsiodes_of_validation`	`4`
`num_lifetimes_for_validation`	`60`
`num_inner_loops_per_update`	`30`
`learning_rate`	`5e-5`
`adam_eps`	`1e-5`
`e_rewards_target_mean`	`0.0001`
`meta_gamma`	`0.9`
`ae.estimation_method`	`'bootstrapping skipping uninfluenced future rewards'`
`ae.bootstrapping_lambda`	`0.85`
`ae.starting_n`	`2200`
`ae.num_n_step_estimates`	`6`
`ae.skip_rate`	`300`
`rnn_input_size`	`32`
`rnn_type`	`lstm`

Hyperparameter	**Value**
`rnn_hidden_state_size`	`128`
`initial_std`	Intrinsic r.=`0.2` Advantages=`1.0`
`ppo.k`	`400`
`ppo.update_epochs`	`12`
`ppo.num_minibatches`	$"0" \approx \left(\frac{\texttt{num inner loop steps}}{\texttt{ppo.k}}\right)$
`ppo.normalize_advantage`	`True`
`ppo.clip_coef`	`0.2`
`ppo.entropy_coef`	`0.0005  Intr.r.ML1=0.003`
`ppo.valuef_coef`	`0.5`
`ppo.clip_grad_norm`	`True`
`ppo.max_grad_norm`	`0.5`
`ppo.target_KL`	`0.01`

Table 2: Hyperparameter configuration for training and evaluating the meta-agents providing intrinsic rewards and advantages.

A.2.2 Hyperparameter Explanations and Further Implementation Details

Architectures: For the policy and critic we used a (64,64) MLP architecture. For the meta-learned networks we used an LSTM with a hidden dimension of 128. Prior to going into the LSTM, the current step's input was processed by two linear(plus activation) layers (128,32) at which all 1 dimensional inputs were concatenated. The LSTM hidden states were processed by a (512) layer for the outer loop critic, a (128) layer for the gaussian distributions standard deviation, and (128,128) layers with a final arctan activation for its mean. All other activations were tanh. When using the learned advantage function the critic was omitted from the inner loop. All implementations where done in PyTorch.

Hyperparameter details: Several of the hyperparameters showed in section A.2.1 are standard in deep RL literature. This section briefly describes those whose influence may be less clear.

- *total_timesteps* refers to the total number of steps collected in inner loops. The length of lifetimes. *num_steps* is the batch size used for PPO updates. Two updates are done during an inner loop.
- *num_epsiodes_of_validation* and *num_lifetimes_for_validation* control how many of the last lifetimes and episodes whithin them are used to validate performance during training. Meta-learning took place until no appreciable performance improvement was observed for at least 200 outer loop updates.
- Throughout meta-training, extrinsic rewards obtained in each inner loop were normalized to keep their average value around *e_rewards_target_mean*. This normalization was done at the level of each environment class. In the ML10 benchmark training, independent averages of extrinsic rewards for each of the 10 problem classes were maintained. This normalization ensures that the meta agent's training signals (extrinsic rewards) stay within the same range during training and assigns similar importance to improvements across different environment classes.
- Hyperparameters starting with *ae* and *meta_gamma* control which variant of Objective 1 is used and how advantages are estimated for outer loop updates. The most conceptually important are *ae.estimation_method* and *meta_gamma*. *ae.estimation_method* controls whether discounts are applied at the episode or step level and which method is used for estimating advantages. Specifically, *'bootstrapping skipping uninfluenced future rewards'* uses episode-wise discounts and estimates advantages with an exponential combination of n-step estimates. This method ignores all extrinsic rewards received by the agent until the generated training signal is used for an update. *meta_gamma* indicates the discount factor for outer loop learning, either step-wise or episodic.

- *initial_std* controls the initial standard deviation of training signals when meta-training.
- Hyperparameters starting with *ppo* for meta-learning have similar but not identical to the standard PPO hyperparameters. PPO updates for the outer loops use truncated backpropagation. Specifically, *ppo.k* controls the number of steps before gradient propagation is truncated. *ppo.num_minibatches* controls the number of processed k-length sequences before a gradient step is taken. The value "0" in the tables indicates a gradient step is taken for each num_inner_loops k-length sequences.
- Hyperparameters whose values were not mentioned in the tables took the default value assigned in Pytorch.

Some of the values that we found to impact performance the most are: *initial_std* , *entropy_coef* , *e_rewards_target_mean*,*num_inner_loops_per_update* and learning rates. Ray was used to run the different inner loops corresponding to an outer loop update in parallel across different processes.

B Appendix Experimental Results

B.1 Meta-learned Policy Parameters

In this section we show results for two popular meta-learning algorithms whose focus is on policy parameters. MAML (its first order version) [17] which meta-learns the initial parameters of a policy and RL^2 [14, 15] which meta-learns a recurrent policy that conditions on all past data. They are applied in the same setting described in section 4. Results can be found in figure 4. It stands out that directly meta-learning policy parameters can lead to informed behaviour from the start of the interaction. MAML showed the best performance of the two methods. Furthermore it attained some generalization when facing non parametric variations. Both methods still have space for improvement and they weren't able to make use of data collected through interaction across episodes to adapt their behaviour (except for MAML in ML1-reach). Further information on their implementation can be found on the work's code repository.

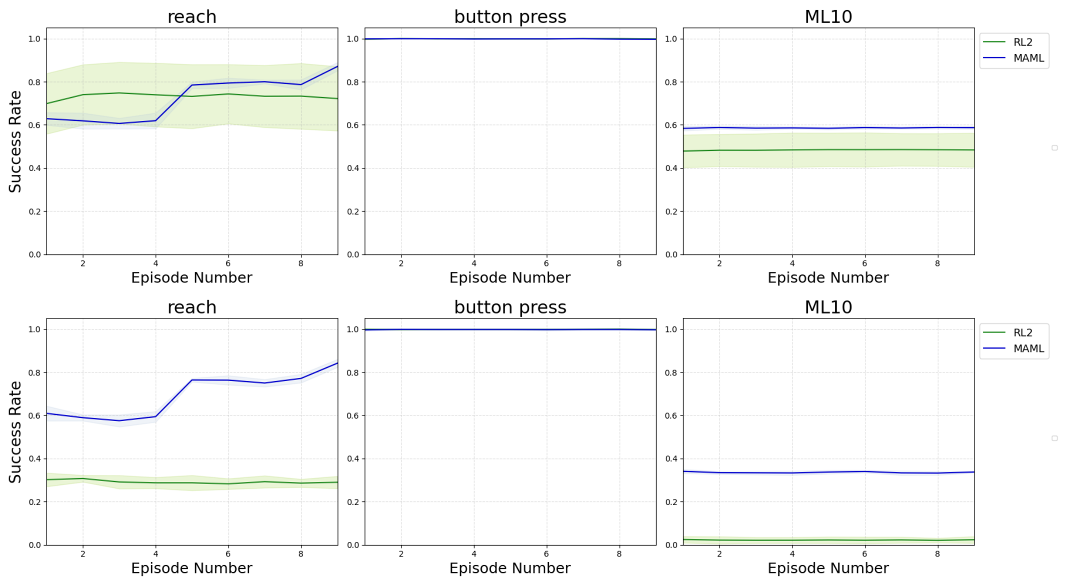

Figure 4: Performance comparison of different methods that meta-learn policy parameters as they interact with tasks from the training (top row) and test set (bottom row). The values and their standard deviations (represented by the shaded region) were obtained as explained in section 4.3. The success rate for RL^2 (green) and MAML (blue) in benchmarks ML1-reach, ML1-button-press and ML10 are shown.