META-GRADIENTS IN NON-STATIONARY ENVIRONMENTS

Jelena Luketina* University of Oxford
Sebastian Flennerhag DeepMind
Yannick Schroecker DeepMind
David Abel DeepMind
Tom Zahavy DeepMind
Satinder Singh DeepMind

ABSTRACT

Meta-gradient methods (Xu et al., 2018; Zahavy et al., 2020) are a promising approach to the problem of adaptation of hyper-parameters in non-stationary reinforcement learning problems. Recent works enable meta-gradients to adapt faster and learn from experience, by replacing the tuned meta-parameters of fixed update rules with learned meta-parameter functions of selected *context* features (Almeida et al., 2021; Flennerhag et al., 2022). We refer to these methods as *contextual meta-gradients*. The context features carry information about agent performance and changes in the environment and hence can inform learned meta-parameter schedules. As the properties of meta-gradient methods in non-stationary environments have not been systematically studied, the aim of this work is to provide such an analysis. Concretely, we ask: (i) how much information should be given to the learned optimizers so as to enable faster adaptation and generalization over a lifetime, (ii) what meta-optimizer functions are learned in this process, and (iii) whether meta-gradient methods provide a bigger advantage in highly non-stationary environments. We find that adding more contextual information is generally beneficial, leading to faster adaptation of meta-parameter values and increased performance. We support these results with a qualitative analysis of resulting meta-parameter schedules and learned functions of context features. Lastly, we find that without context, meta-gradients do not provide a consistent advantage over the baseline in highly non-stationary environments. Our findings suggest that contextualising meta-gradients can play a pivotal role in extracting high performance from meta-gradients in non-stationary settings.

1 INTRODUCTION

Meta-gradient approaches to learning adaptive optimizers are a promising complement to hand-engineering gradient-based optimizers in reinforcement learning (RL). By adapting relevant optimization hyperparameters or the entire update rule to the current domain, they often outperform well-tuned gradient-based optimizers (Schraudolph, 1999; Mahmood et al., 2012; Andrychowicz et al., 2016; Xu et al., 2018). The adaptability of meta-learned optimizers is particularly relevant for non-stationary environments: as the agent continues learning during its lifetime, appropriate hyperparameter values for an optimizer are likely to change over time and can be impossible to determine in advance (Parker-Holder et al., 2022). Despite this promise, the performance and properties of meta-gradients in the context of reinforcement learning in non-stationary environments have not been systematically studied, which is the focus of our work. The question we focus on is how does the information provided to the meta-optimizer as well as the rate of environment non-stationarity effect the performance and properties of meta-gradients.

Most of the prior work in this area can be classified within the framework of white and black box optimization (see Related Work in Xu et al. (2020) for a more detailed classification of prior work). Black-box methods express the entire update rule as a rich parametric function, typically a recurrent neural network, and learn the parameters of this function in an end-to-end manner (Xu et al., 2020; Oh et al., 2020; Kirsch et al., 2021). In contrast, white-box methods tune the hyperparameters of the optimization algorithm (Mahmood et al., 2012; Xu et al., 2018; Zahavy et al., 2020). We

*Work done during an internship at DeepMind. Contact: jelena.luketina@cs.ox.ac.uk

limit the scope of our analysis to white-box meta-gradient methods for self-tuning hyperparameters, which have to date shown the greatest empirical gains in RL (Xu et al., 2018; Zahavy et al., 2020; Flennerhag et al., 2022).

Since white-box methods are almost completely memory-less and only tune several hyperparameters based on their local influence on agent performance, standard white-box meta-gradients are only capable of tracking good solutions (Sutton et al., 2007) and lack the ability to learn and benefit from past experience. These problems are partly addressed in some recent works in reinforcement (Flennerhag et al., 2020) and supervised (Almeida et al., 2021) learning, which extend the standard white-box formulation by replacing learned hyper-parameters with learned functions of hand-picked *context* features. Broadly speaking, context features can be any low-dimensional statistics of optimization, agent, or environment that carry information about suitable hyperparameter schedules. Examples of such context features in RL are reward histories, temporal difference errors, and task beliefs. We refer to this approach as *contextual meta-gradients*. While sharing some similarities with black-box meta-gradients, contextual meta-gradients differ in two important ways: the space of learned updated rules is limited (the update rule can only be changed through hyperparameters) and they have access to different kind of information (selected context features instead of history of gradients or parameters).

In this paper, we hypothesize that: (1) the addition of context to meta-gradients is particularly well-suited for non-stationary environments with some repeated structure, as it enables the optimizer to generalize from previously seen contexts and instantly pick good hyper-parameter values; (2) the advantage of all meta-gradients methods (with or without context) over well-tuned fixed hyperparameter schedules increases with the rate of environment non-stationarity, as different environment conditions may require very different learning hyperparameter values.

To examine the first hypothesis, we compare the performance of meta-gradients with and without contextual information across several environments. Since the addition of contextual information can enable the meta-learner to learn a hyper-parameter schedules to generalize across learning contexts, we look at how increasing the context richness (i.e. amount of information given to the meta-learner) effects the performance. We find that adding more information to the meta-optimizer most of the time helps training (although some context are much more useful than others), and in some cases, it is necessary for learning quickly adapting meta-parameter schedules. We complement these findings by examining learned schedules and functions of context, which demonstrate faster adaptation of hyperparameter values and emergence of meaningful functions of context. Lastly, we investigate the adaptation ability of meta-gradients as the rate of environmental non-stationarity increases, with a particular interest in potential advantages of context in highly non-stationary environments. We find that *without* contextual information, meta-gradients provide small to inconsistent advantage over fixed hyperparameter values in highly non-stationary environments. On the other hand, meta-gradients *with* contextual information provide a much more consistent advantage.

2 Related Work

The focus of our paper is on white-box methods for learning adaptive optimizers (Bengio, 2000; Maclaurin et al., 2015). These methods tune the meta-parameters (i.e. tunable subset of hyperparameters) of the learning update by computing the gradient of the outer loss with respect to meta-parameters, where the outer loss depends on meta-parameters both directly and through the history of last K parameter updates. In RL, this approach has been used to tune various optimization hyperparameters, including discount and bootstrap parameters (Xu et al., 2018), off-policy corrections (Zahavy et al., 2020), auxiliary rewards and tasks (Zheng et al., 2018; Veeriah et al., 2019; 2021) and weights of rewards in return estimates (Wang et al., 2019).

Among these methods, Flennerhag et al. (2022) and Almeida et al. (2021) propose learning meta-parameters as functions of context features, with the application in reinforcement and supervised learning respectively. While the focus of Flennerhag et al. (2022) is on reducing myopia and conditioning problems of meta-gradients by developing an improved outer loss, in their experiments on non-stationary environments, they parameterize meta-parameters as a function of contextual features (in their case, reward statistics). However, they do not study the importance of including contextual information or provide comparison to alternative contextual features. In Almeida et al. (2021), learning meta-parameters as function of features is the primary focus. In contrast to most prior work in white-box meta-gradients, Almeida et al. (2021) formulate optimization of meta-parameters as a reinforcement learning problem. The meta-parameter function is a policy modifying the current values of corresponding meta-parameters, trained on a distribution of source tasks with a designed

reward function. The context features are selected to facilitate transfer of learned optimizers to the target tasks. In contrast to our work, their focus is on achieving high performance on supervised learning tasks instead of analysis of meta-gradients in non-stationary RL environments.

For related work in black-box meta-gradients and the use of context outside of work in meta-gradients, see Appendix A.1.

3 BACKGROUND

We follow the meta-gradient literature and make a distinction between the parameters θ (e.g. weights of the policy and value networks) and the meta-parameters η (e.g. learning rates, discounts, weights of regularization losses). The parameters θ are trained by minimizing an inner loss function $\mathcal{L}_\eta^{(\text{inner})}(\theta, \mathcal{D})$ parameterized by the meta-parameters η, whereas the meta-parameters η are trained to minimize the outer loss $\mathcal{L}^{(\text{outer})}(\eta, \mathcal{D})$, where in both cases $\mathcal{D}$ refers to the rollout data used to estimate the loss. The parameters and meta-parameters are generally optimized at two different time-scales; we denote by i the inner loop and by k the outer loop iteration step.

The outer loss objective depends on the meta-parameters η through their influence on the learned parameters θ. This dependence is given by the update rule: $\theta_{i+1} = f(\theta_i, \eta_k, \mathcal{D}_i)$, where $\mathcal{D}_i$ refers to the data used to compute the losses at i-th iteration. For example, for stochastic gradient descent, $\theta_{i+1} = \theta_i - \text{lr}\nabla_\theta \mathcal{L}_\eta^{(\text{inner})}(\theta)$, which is parameterized by η through $\mathcal{L}_\eta^{(\text{inner})}(\theta)$. Since backpropagating through the entire history of updates is too computationally demanding and may not provide a good gradient estimate due to non-stationarity intrinsic to RL, the meta-gradients are truncated to the last K inner loop updates, which we refer to in experimental section as *meta rollout length*. The outer losses can be computed on a new rollout data $\mathcal{D}_k$ or the rollout data used in the inner loop $\mathcal{D}_i$. A more detailed description of common inner and outer objectives can be found in the Appendix A.2.

4 CONTEXTUAL META-GRADIENTS

Contextual meta-gradients extend white-box meta-gradients by replacing the meta-parameter variables η_k in the update rule with parameterized functions $\eta_k(\cdot)$ of context features c_i: $f(\theta_i, \eta_k(c_i), \mathcal{D}_i)$. The meta-parameters η of the meta-parameter function $\eta(\cdot)$ are learned by optimizing the outer loss $\mathcal{L}^{(\text{outer})}(\eta)$. To avoid the overloaded notation and terminology, throughout this paper, we use η_k and the term meta-parameter to denote two related quantities: the parameters of the meta-parameter network, as well as the predictions of the meta-parameter network, at outer iteration k. It will be clear from context which one we are referring to. Also note that depending on how the context features are shared, this change allows for different meta-parameter predictions between individual inner updates or even individual samples contributing to the inner loss. Similar ideas were explored in Flennerhag et al. (2022) and Almeida et al. (2021).

The context features can be any learning or environment statistics that capture information relevant for optimal meta-parameter schedules. We do not assume that context features will be repeated during a lifetime, however choosing context features that do repeat should provide contextual meta-gradients with a greater advantage over non-contextual meta-gradients. If such repetition exists, meta-parameter functions can learn and generalize schedules from previously seen contexts. In our experiments, the features were chosen to indicate changes in task at hand and the agent performance (e.g. statistics of rewards, TD-errors, values), all of which may require different meta-parameter values. Lastly, we note that gradients do not propagate through the context features.

5 ENVIRONMENTS

For our analysis, we use two non-stationary environments. The agent interacts with these environments via a single-stream of experience via the standard RL formulation (there is only one environment and not multiple copies of it, as is common in distributed RL frameworks). The simplicity of the environments allows for a fine control over the sources of non-stationarity and makes it easier to interpret learned meta-parameter schedules and functions. More information about environments can be found in Appendix A.4.

Two Colors. We start with a gridworld environment introduced in Flennerhag et al. (2022), where the agent is tasked with picking up one of the two items. Picking up one of the items results in reward

+1 or -1, after which the agent and two objects are randomly re-spawned. Which object carries positive reward changes every 100,000 steps. The agents used in our experiments are memory-less, hence the optimal policy has to be re-learned after each task switch, which requires increasing the rate of exploration after the task switches.

Switching MDPs. In our second non-stationary setting, the agent interacts with a sequence of grid worlds, where a new grid world is sampled every 100,000 steps from a set of N grid worlds. The reward function and transition function of each grid world are randomly generated.

We use this environment to study how changes in the environment dynamics changes, in addition to the reward function, affect meta-gradient methods. Additionally, in contrast to the Two Colors environment, two consecutive tasks may not be as different from each other and contain some positive transfer. This implies that in some cases, the optimal exploration strategy upon a task change may not be to explore too much.

We experimented with two variants of Switching MDPs with different number of grid-worlds N. In the first case N=4 MDPs so that the agent is repeatedly exposed to the same tasks, and can improve its learning over time. In the second case, N=1000 MDPs, and the agent is very unlikely to experience the same task twice, requiring constantly learning almost from scratch in each MDP. This case is interesting, because we can study the generalization of contextual meta-gradients to unseen contexts. As tasks effectively do not repeat, distribution of context features that are given to the meta-parameter function after each task switch is much more irregular. For example, we do not observe regular periodic drops and increases in mean reward as we do in Two Colors.

6 EXPERIMENTS

We designed the experiments with the goal of answering the following questions about the behaviour of meta-gradients in non-stationary environments:

Do meta-gradients benefit from contextual information? We hypothesise that ability to learn meta-parameter schedules as functions of contextual information will enable faster adaptation in non-stationary environments, as the optimizer can leverage knowledge from previously seen contexts and instantly use good meta-parameter values without needing to slowly tune them. We compare the performance of contextual meta-gradients and baselines in Section 6.1.

What functions of context are learned in this process? To explain the performance difference between methods, we examine the meta-parameter schedules during training and the learned meta-parameter functions of context in Section 6.2.

What should the contextual information be? The choice of context features could be essential for learning schedules that generalize over the training. Here we look into increasingly rich contexts: what is the effect of adding particular candidate contexts and can too much information be detrimental for generalization? We shed light on these questions in Section 6.3.

How does the performance of meta-gradients depend on the rate of non-stationarity? As the rate of change of the environment increases, we hypothesized that the ability to adapt the meta-parameters becomes more important. Furthermore, the addition of context to meta-gradients should lead to further advantages, as the they enable faster adaptation. The advantage of meta-gradients under different rates of non-stationarity is examined in Section 6.4.

EXPERIMENTAL AXES

To ensure that the conclusions we draw are robust, we run the experiments along several axes: (i) agents in inner loop, (ii) outer loop objectives, (iii) context features.

Agents. We use two different kinds of RL agents: Actor-Critic (AC) (Sutton et al., 1999) and $Q(\lambda)$ (Peng & Williams, 1994). The parameters of an AC agent are updated every 16 environment steps, and those of $Q(\lambda)$ agent at each step. If using AC agent, we tune the coefficient (i.e. weight) of entropy loss in the inner loss, whereas if using $Q(\lambda)$ agent, we tune the ϵ parameter of ϵ-greedy exploration. The details of the agent architectures and objectives can be found in Appendix A.5.

Outer Loop Objectives. In experiments combining meta-gradients (MG) with AC agents, the outer loss is a sum of policy loss $\mathcal{L}_\pi$ and a target entropy loss $\mathcal{L}_{\text{ent}}$ (the coefficient of this entropy loss is a fixed hyperparameter $\alpha_{\text{ent}}^{(outer)}$):

Table 1: Two Colors with AC (**left**) and $Q(\lambda)$ Agents (**right**). Mean and standard deviation of total return after 10M steps (10 seeds). **Left:** Only meta-gradient methods *with* context features significantly outperform the AC baseline. **Right:** All meta-gradient methods significantly outperform the $Q(\lambda)$ baseline, with the highest performance obtained by meta-gradients with Reward context features.

	Context Features		
Method	None	Reward	Rich
AC	1.24 (0.09)	N/A	N/A
AC-MG	1.29 (0.08)	1.58 (0.06)	1.62 (0.10)
AC-BMG	1.32 (0.08)	1.74 (0.03)	**1.79 (0.07)**

	Context Features	
Method	None	Reward
Q(λ)	0.58(0.07)	N/A
Q(λ)-BMG	1.78(0.03)	**1.93(0.03)**

$$\mathcal{L}_{\text{MG}}^{(\text{outer})} = \mathcal{L}_{\pi} + \alpha_{\text{ent}}^{(\text{outer})} \mathcal{L}_{\text{ent}}. \tag{1}$$

If using Bootstrapped Meta-Gradients (BMG) (Flennerhag et al., 2022), the outer loss is the KL divergence between the target policy $\pi_{\hat{\theta}}$ and K-step bootstrap π_{θ_K}:

$$\mathcal{L}_{\text{BMG}}^{(\text{outer})} = KL(\pi_{\hat{\theta}} || \pi_{\theta_K}), \tag{2}$$

where the target policy is the policy reached after $K + L - 1$ steps of inner loop optimization. For a longer description of BMG loss and how it can be made differentiable with respect to ϵ when used with $Q(\lambda)$ agents, see Appendix A.3.

Context Features. We compare two kinds of contextual features, simple features referred to as *Reward* (only using reward statistics) and more complex features referred to as *Rich* (using reward, TD-error and value statistics). If the context is *Reward*, each context feature is either a history of the last H observed rewards ($Q(\lambda)$ agent) or a history of mean rewards observed in each of the last H rollouts (AC agent). If the context is *Rich*, we compute the same history for reward, TD-error and values. In AC experiments, rich context also includes a history of last H standard deviations in addition to means. We use $H = 10$ in experiments with AC agents, and $H = 100$ in experiments with $Q(\lambda)$ agents. Each context feature is normalized and scaled to be in the [-1, 1] range, before being concatenated and passed down to the learned meta-parameter function. More detailed description of context features used in our experiments can be found in Appendix A.6.

In each experiment, **we report only the results for the best hyper-parameter setting**. For the baselines (which do not adapt the meta-parameters), we include the meta-parameter of interest into the hyperparameter sweep. For MG objectives, we sweep over the meta learning rate, meta rollout length (K) and target entropy loss coefficient ($\alpha_{\text{ent}}^{(\text{outer})}$). For BMG methods, we sweep over meta learning rate, meta rollout length (K) and target rollout length (L). For fairness, the size of hyperparameter sweeps is the same for all meta-gradient methods. For more details on the hyper-parameter sweeps and values, see Appendix A.5.

6.1 Do Meta-Gradients Benefit from Adding Contextual Information?

We start the experiments by testing how adding contextual information to meta-gradients effects training. If our hypothesis is true, we expect to see the advantages of adding contextual information across different agents and outer loop objectives.

Two Colors: Actor Critic Agent. We first look at the AC agents trained on Two Colors (described in Section 5). In Table 1 (left), we report the mean and standard deviation of the total reward after 10M environment steps.

We find that all meta-gradient methods outperform the baseline with fixed meta-parameters (AC) and, consistent with reports in Flennerhag et al. (2022), BMG performs better across methods. More importantly, the addition of context (see columns with Context Features set to Reward or Rich) is crucial for obtaining significant improvement over the baseline, with the richer features performing slightly better.

Two Colors: $Q(\lambda)$ Agent. Next, we look at the $Q(\lambda)$ agents on Two Colors. In Table 1 (right), we again report the mean and standard deviation of total returns after 10M environment steps. We tune the ϵ parameter of ϵ-greedy exploration with only BMG objective, as the updated value function in this case is not a differentiable function of ϵ and consequently can not be straightforwardly optimized with regular MG objectives.

Table 2: Switching MDPs with $Q(\lambda)$ Agent. Mean and standard deviation of total return after 20M steps (10 seeds). Contextual information was necessary to gain a significant improvement over the baseline with meta-gradients. The improvement is greater when the number of different MDPs (i.e. variety of tasks experienced during a lifetime) is greater.

	Number of MDPs	
Method	4	1000
$Q(\lambda)$	14.77 (1.78)	12.03 (0.96)
$Q(\lambda)$-BMG	14.79 (1.56)	11.71 (0.71)
$Q(\lambda)$-BMG-Reward	15.00 (2.21)	13.36 (0.68)
$Q(\lambda)$-BMG-Rich	**16.03 (1.40)**	**13.63 (0.64)**

We find that meta-gradients significantly outperform the non-adaptive baseline and the addition of context further boosts the performance.

Switching MDPs: $Q(\lambda)$ **Agent**. Next, we look at $Q(\lambda)$ agent trained on Switching MDPs. In Table 2, we report the mean and standard deviation of total rewards after 20M environment steps. We report only the experiments with $Q(\lambda)$ agents since AC agents were performing very poorly on this environment. Again, we tune only the ϵ parameter. For these experiments, we also vary the number of different MDPs available in the environment (see Section 5 on meaning and importance of these environment variations).

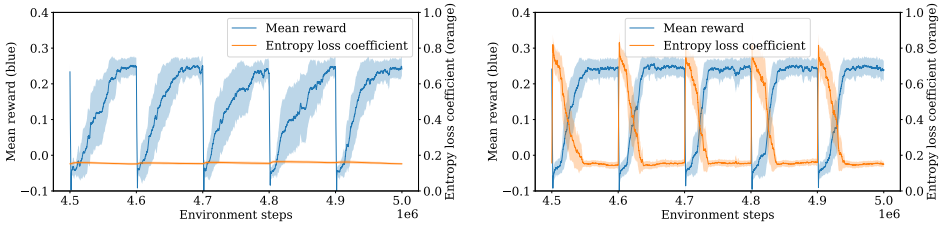
We compare the results under the two environment variants of interest (4 and 1000 MDPs) in Table 2. The addition of context was necessary to obtain significant improvements with meta-gradients, with Rich context resulting in the biggest improvement. The improvement is also much more significant in the regime where the MDPs effectively do not repeat (1000 MDPs). Because the consecutive tasks will typically be more similar compared to Two Colors, we hypothesize there is less need to re-learn in the regime with a small number of tasks (4 MDPs), requiring less adaptation of meta-parameters.

6.2 What Meta-Parameter Schedules and Functions are Learned?

In this section, we wish to examine the predictions of the learned meta-parameter functions throughout training. For the ease of analysis, we focus on BMG with Reward context features (BMG-Reward) while only tuning one meta-parameter.

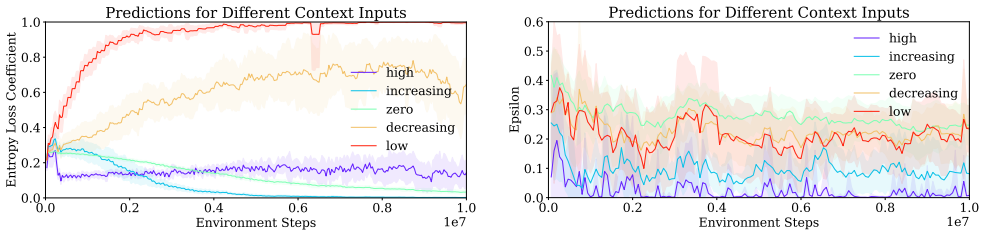
Learned Schedules. We look at the relationship between the learned meta-parameter schedule and the observed rewards for AC agents with BMG trained on Two Colors. In Figure 1a, meta-gradients do not rely on context features, and in Figure 1b, meta-gradients use Reward context features. The two curves in both figures are: entropy loss coefficient during training (orange curve) and mean reward over rollout (blue curve).

When meta-gradients do not have access to reward context (Figure 1a), the mean reward takes longer to recover after the drop following each task switch. Furthermore, entropy coefficient is almost constant. Note that this can not be explained by too low meta learning rate: the results shown here



(a) BMG. **(b)** BMG-Reward.

Figure 1: Entropy loss coefficient (orange) vs. mean reward (blue) during training for: (a) AC with BMG, (b) AC with BMG and reward context. We report values at each environment step in the middle of training, averaged over 10 seeds. The error margins represent 95% confidence intervals. The drops in mean reward at regular intervals correspond to task switches. Without access to contextual information, meta-gradients learn an almost constant meta-parameter schedule. By adding reward context, learned meta-parameter schedule strongly responds to drops in mean reward.



(a) Two Colors with AC-BMG-Reward. **(b)** Switching MDPs (1k) with $Q(\lambda)$-BMG-Reward

Figure 2: Predicted values of exploration meta-parameters for five qualitatively different reward context features as inputs to the meta-parameter function, as measured during training in: (a) Two Colors with AC-BMG-Reward, (b) Switching MDPs (1000 MDPs) with $Q(\lambda)$-BMG-Reward. Each curve is averaged over 10 random seeds with the error margins representing one standard deviation. In (a), the α_{ent} is predicted when the rewards are the lowest, and the lowest when the rewards are increasing. In (b), the highest ϵ is predicted for lowest or zero rewards, and the lowest for highest rewards.

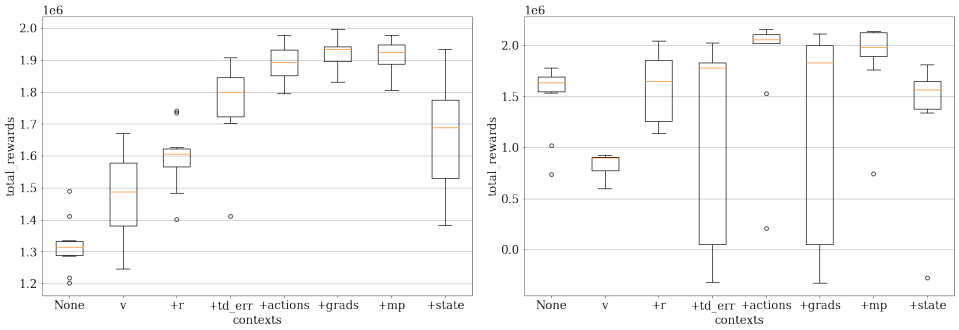
are under the best hyper-parameter configuration and the best performing meta learning rates were never the highest values in the sweep. In contrast, when the reward context is added (Figure 1b), the entropy coefficient rapidly increases after each task switch to allow the agent to explore. As a result, the mean rewards of the agent are better as we observed in the previous section, suggesting that the addition of context is crucial for obtaining fast adaptation.

Learned Functions. Next, we look at how the context to meta-parameter mapping itself changes during a lifetime. Our methodology is inspired by visualization of model trajectories in Erhan et al. (2010). Due to low dimensionality of inputs and outputs of meta-parameter function, we can design a small number of representative context inputs and for each, track the corresponding meta-parameter prediction during training. Note that the meta-parameter function is not trained on these context inputs, we just log the output for each while the meta-parameter function is trained as usual.

The five context inputs were designed to represent qualitatively different learning situations: constantly high context values (*"high"*), monotonically increasing from lowest to highest value (*"increasing"*), constant zero (*"zero"*), monotonically decreasing between the highest and lowest value (*"decreasing"*) and constantly low context (*"low"*). For example, for rewards bounded in $[-1, 1]$ and $H = 3$, the context inputs are: "high" $= [1, 1, 1]$, "increasing" $= [-1, 0, 1]$, "zero" $= [0, 0, 0]$, "decreasing" $= [1, 0, -1]$ and "low" $= [-1, -1, -1]$. In our experiment, $H = 10$ and the range is $[-1, 1]$ due to pre-processing of context features. Note that some of the context inputs have the same mean (e.g., *"increasing"*, *"zero"* and *"decreasing"* all have mean zero), however they capture qualitatively different behaviors. For example, the *"decreasing"* input corresponds to a context likely observed just after switching the task, while the *"increasing"* probe is likely observed as the agent starts improving in a new task.

We visualize the learned meta-functions of AC agent trained with BMG-Reward on Two Colors in Figure 2a, and those of $Q(\lambda)$ agent trained with BMG-Reward on Switching MDPs (1000 MDPs variant) in Figure 2b. First, we can observe that different inputs results in very different meta-parameter predictions and that the meta-parameter functions seem to converge during training (the small local changes are likely due to non-stationary training distribution), hence addition of context enables convergence instead of just tracking good meta-parameter values as for vanilla MG (Sutton et al., 2007).

Next, we look at the values of predicted meta-parameters for each of the five context inputs. We can observe that the differences in predictions are sensible: in Figure 2a, the entropy loss coefficient is highest when the rewards are at low or decreasing, and the lowest when the rewards are high or decreasing; whereas in Figure 2b, the exploration is highest when the rewards are low, decreasing or zero and lowest when the rewards are high. The predictions are different for context inputs with the same mean value ("increasing", "decreasing", "zero"), indicating learned function responds to more complex patterns than just mean values of features. Lastly we note that for Switching MDPs, we see more variability in learned functions across different training seeds. This is likely due to a more complex interaction between encountered tasks and stochasticity in generating and sampling tasks (i.e. how much transfer there is between consecutive tasks will depend on which two tasks are sampled).



(a) AC-BMG (α_{ent}) **(b)** AC-BMG (α_{ent}. α_{L2})

Figure 3: Performance of meta-gradients as a function of increased context richness: (a) AC-BMG with tuned entropy loss coefficient, (b) AC-BMG with tuned entropy and L_2 loss coefficients. Total reward at 10M environment steps (10 seeds). Starting without contextual information (None), in each column, from left to right, we add the statistics of the following quantities as features to the context: value (v), reward (+r), TD-error (+td_err), action probabilities (+actions), cosine distance between gradients (+grads), previous values of meta-parameters (+mp) and state (+state). With some exceptions, adding more information leads to increase in mean performance.

6.3 How Important is the Choice of Context?

In the following set of experiments, we inspect how making the context more rich effects performance. For this goal, we focus on Two Colors with the AC agent and BMG objective. We consider two variants; one that tunes the entropy coefficient (Figure 3a) and one that tunes the coefficients of both the entropy and the L_2 loss (Figure 3b). The L_2 loss in our experiments is defined over the weights of both policy and value functions. See Appendix A.5 for a longer description of the terms used in the inner loss.

The x-axis of each figure corresponds to the degree of richness of the context. On the left, we start with no context, and then add one-by-one the statistics of following quantities as context features: value, reward, TD-error, action probabilities, cosine distance between last two gradients, past meta-parameter predictions and state statistics. For each of these quantities (except cosine distance and past meta-parameter predictions), the features are a history of means and standard deviations calculated over the last H rollouts. For cosine distance and past meta-parameter predictions, the features are just a history of their last H values. Since the number of features is large, we decrease the history length H from 10 to 4. If we were to use $H = 10$, the dimension of richest context input would be 660, which could make the optimization of meta-function too challenging (see Appendix A.6 for description and dimensions of each feature).

The only two context features that seem to harm the performance (Figure 3b) are value features and state features. The drop seen at the end by adding state visitation statistics is likely due to overfitting. State features have much higher dimensionality compared to other features (see Appendix A.4 and A.6 for more detailed description of state features), yet they are not informative of the agent performance and hence good meta-parameter values. We do not observe as much overfitting for other features. Generally speaking, richer features lead to better performance as there are more potentially useful sources of signal for the meta-parameter function.

6.4 How do Meta-Gradients Perform under Different Rates of Non-Stationarity?

Lastly we look at how the degree of non-stationarity in the environment effects the performance of meta-gradients. We control the degree of non-stationarity by changing how often the tasks switch – shorter change periods correspond to higher rates of non-stationarity. In all of the Figures included in this section, the medians and interquartile ranges are computed over 10 random seeds.

Two Colors. In this experiment, we compare the relative improvement of different meta-gradient methods over the AC baseline for various non-stationarity rates (Figure 4).

The relative improvement is defined as percent increase or decrease in performance (as measured in total reward after 10M steps) over the mean performance of the baseline (with the same non-stationarity rate). The absolute (non relative) values of all the methods can be found in the Appendix

A.7. We use BMG as an outer loop objective and tune entropy rate coefficient. The compared methods are: BMG (no context), BMG-Reward and BMG-Rich. As before, we report only the results for the best hyper-parameter configurations.

We find that when the environment changes too rapidly, at around 50,000 steps, the performance of meta-gradients deteriorates possibly due to their myopic nature. We expected to find that improvement brought on by meta-gradients is greater when the environment changes more rapidly. This expectation was met up to a point: BMG with a rich context were less affected in addition to outperforming context-less meta-gradients for any rate of change, indicating that providing more information can provide enough signal to enable learning even in this regime.

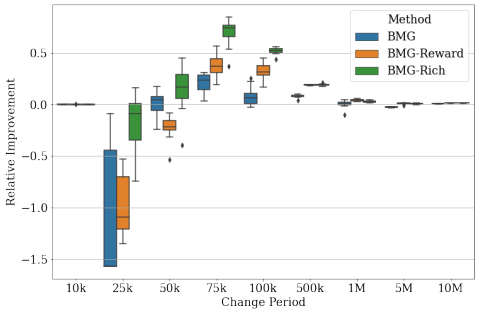


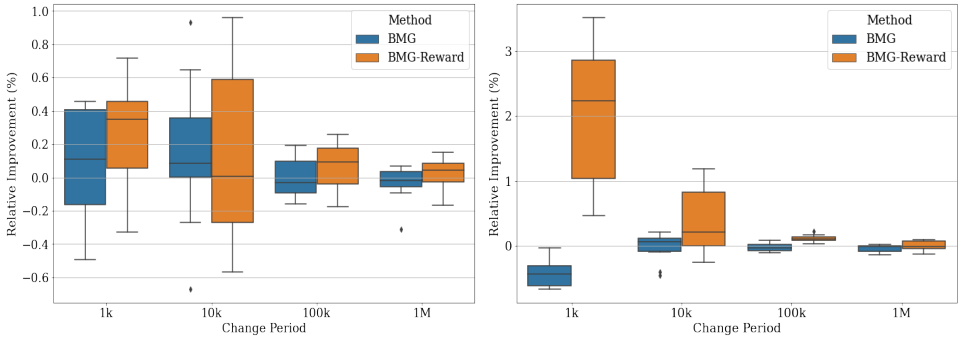
Figure 4: Two Colors: Relative improvement over the fixed meta-parameter AC baseline and under different rates of non-stationarity. The regime in which meta-gradients provide a significant advantage over the baseline is the greatest for meta-gradients with rich context [50k, 500k]. All meta-gradient fail to beat the baseline when the rate of non-stationarity is too high (25k).

Switching MDPs. Figure 5a and Figure 5b present the relative improvement of meta-gradients over the $Q(\lambda)$ baseline in the Switching MDPs environment (for $N = 4$ and $N = 1000$ respectively). Absolute (non relative) values can be found in the Appendix A.7.

We find that the use of context becomes more effective as we increase the rate of environment non-stationarity and the number of different tasks, where fast adaptation of meta-parameters becomes more relevant.

7 Conclusions

We studied the performance and properties of white-box meta-gradients in non-stationary environments. To study the effect of adding contextual information to the learned optimizer, we focused on formulations of meta-gradients where the learned meta-parameter values are functions of selected context features. We found that adding more contexual information is almost always beneficial for lifetime performance. Inspection of learned meta-parameter schedules and functions provides evidence of faster adaptation for meta-gradients with contextual information and convergence of meta-parameter functions over training. When looking at the effect of increasing the rate of non-stationarity, we find that the meta-gradients without context, in contrast to meta-gradients with context, do not offer a large consistent advantage over fixed meta-parameter schedules. An interesting avenue for future research are studies of contextual meta-gradients in non-stationary environments with less repeatability of contexts and continual supervised learning.



(a) 4 MDPs **(b)** 1000 MDPs

Figure 5: Relative improvement with meta-gradients over the $Q(\lambda)$ baseline under different rates of environment non-stationarity: (a) for Switching MDPs with 4 MDPs, (b) Switching MDPs with 1000 MDPs. Meta-gradients with Reward context provide a more reliable performance boost, with the biggest improvement when the rate of non-stationarity and the number of different tasks are high.

REFERENCES

Diogo Almeida, Clemens Winter, Jie Tang, and Wojciech Zaremba. A generalizable approach to learning optimizers. *arXiv preprint arXiv:2106.00958*, 2021.

Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to Learn by Gradient Descent by Gradient Descent. In *Advances in Neural Information Processing Systems*, 2016.

Stav Belogolovsky, Philip Korsunsky, Shie Mannor, Chen Tessler, and Tom Zahavy. Inverse reinforcement learning in contextual mdps. *Machine Learning*, 110(9):2295–2334, 2021.

Yoshua Bengio. Gradient-Based Optimization of Hyperparameters. *Neural computation*, 12(8): 1889–1900, 2000.

Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pretraining help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 201–208. JMLR Workshop and Conference Proceedings, 2010.

Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-Learning with Warped Gradient Descent. In *International Conference on Learning Representations*, 2020.

Sebastian Flennerhag, Yannick Schroecker, Tom Zahavy, Hado van Hasselt, David Silver, and Satinder Singh. Bootstrapped meta-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=b-ny3x071E5.

Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving Generalization in Meta Reinforcement Learning Using Learned Objectives. 2020.

Louis Kirsch, Sebastian Flennerhag, Hado Philip van Hasselt, Abram L. Friesen, Junhyuk Oh, and Yutian Chen. Introducing symmetries to black box meta reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-Based Hyperparameter Optimization Through Reversible Learning. In *International conference on machine learning*, pp. 2113–2122. PMLR, 2015.

Ashique Rupam Mahmood, Richard S Sutton, Thomas Degris, and Patrick M Pilarski. Tuning-free step-size adaptation. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2121–2124. IEEE, 2012.

Aditya Modi, Nan Jiang, Satinder Singh, and Ambuj Tewari. Markov decision processes with continuous side information. In *Algorithmic Learning Theory*, pp. 597–618. PMLR, 2018.

Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver. Discovering Reinforcement Learning Algorithms. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, et al. Automated reinforcement learning (autorl): A survey and open problems. *arXiv preprint arXiv:2201.03916*, 2022.

Jing Peng and Ronald J. Williams. Incremental Multi-Step Q-Learning. In *International Conference on Machine Learning*, 1994.

Nicol N. Schraudolph. Local Gain Adaptation in Stochastic Gradient Descent. In *International Conference on Artificial Neural Networks*, 1999.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, volume 99, 1999.

Richard S Sutton, Anna Koop, and David Silver. On the role of tracking in stationary environments. In *Proceedings of the 24th international conference on Machine learning*, pp. 871–878, 2007.

Vivek Veeriah, Matteo Hessel, Zhongwen Xu, Janarthanan Rajendran, Richard L Lewis, Junhyuk Oh, Hado P van Hasselt, David Silver, and Satinder Singh. Discovery of useful questions as auxiliary tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

Vivek Veeriah, Tom Zahavy, Matteo Hessel, Zhongwen Xu, Junhyuk Oh, Iurii Kemaev, Hado P van Hasselt, David Silver, and Satinder Singh. Discovery of options via meta-learned subgoals. *Advances in Neural Information Processing Systems*, 34, 2021.

Yufei Wang, Qiwei Ye, and Tie-Yan Liu. Beyond exponentially discounted sum: Automatic learning of return function. *ArXiv*, abs/1905.11591, 2019.

Zhongwen Xu, Hado P. van Hasselt, and David Silver. Meta-Gradient Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2018.

Zhongwen Xu, Hado P van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver. Meta-gradient reinforcement learning with an objective discovered online. *Advances in Neural Information Processing Systems*, 33:15254–15264, 2020.

Tom Zahavy, Zhongwen Xu, Vivek Veeriah, Matteo Hessel, Junhyuk Oh, Hado P van Hasselt, David Silver, and Satinder Singh. A Self-Tuning Actor-Critic Algorithm. *Advances in Neural Information Processing Systems*, 33, 2020.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On Learning Intrinsic Rewards for Policy Gradient Methods. *Advances in Neural Information Processing Systems*, 2018.

A Appendix

A.1 Related Work

An alternative approach to white box methods for learning adaptive optimizers are the black box methods, which parametrize the entire update rule as a neural network, and learn it from scratch by training on a distribution of supervised (Andrychowicz et al., 2016) or reinforcement learning (Kirsch et al., 2020; Oh et al., 2020; Kirsch et al., 2021) tasks. In RL, most of these methods require training over multiple tasks or lifetimes, which makes them not directly applicable to our setting of interest. The most relevant work is Xu et al. (2020), which develops a black-box method that trains an optimizer over a single lifetime. In black-box methods, the inputs to the learned optimizer functions are typically a history of parameters and gradients, but more similar to context features explored in our work, they can also include the entire rollout trajectories (Xu et al., 2020; Oh et al., 2020). However, these works do not explore the importance of selecting inputs for the learned optimizer and its effects on training in non-stationary environments.

Another related line of work is that of Contextual MDPs (Hallak et al., 2015; Modi et al., 2018; Belogolovsky et al., 2021). In this setting, contexts, which define the reward and transition kernel, are sampled from a distribution. This formulation allows the study generalisation, e.g., we can learn a mapping from contexts to MDPs and ask how well should this mapping generalize to unseen contexts; see (Kirk et al., 2021) for a recent survey on the topic. Our work, on the other hand, does not make such an assumption about the world and instead focuses on a single, but non-stationary MDP. It might be possible to think about some specific non-stationary MDPs as special cases of contextual MDPs, but we leave this exciting direction for future work.

A.2 Meta-Gradients Objectives

Recall from Section 3 that the parameters θ are trained by minimizing an inner loss function $\mathcal{L}_\eta^{(\text{inner})}(\theta, \mathcal{D})$ parameterized by the meta parameters η, whereas the meta-parameters η are trained to minimize the outer loss $\mathcal{L}^{(\text{outer})}(\eta, \mathcal{D})$, where in both cases $\mathcal{D}$ refers to the rollout data used to

estimate the loss. The parameters and meta-parameters are generally optimized at two different time-scales; we denote by i the inner-loop and by k the outer-loop iteration step.

Most works in RL rely on entropy-regularized actor-critic algorithms (Xu et al., 2018; Zahavy et al., 2020), where the inner loss objective $\mathcal{L}^{(\text{inner})}(\theta, \mathcal{D})$ consists of the following three terms (policy, value and entropy loss respectively):

$$\mathcal{L}^{(\text{inner})}(\theta, \mathcal{D}) = \alpha_\pi \mathcal{L}_\pi(\theta, \mathcal{D}) + \alpha_v \mathcal{L}_v(\theta, \mathcal{D}) + \alpha_{\text{ent}} \mathcal{L}_{\text{ent}}(\theta, \mathcal{D}) \tag{3}$$

$$\mathcal{L}_\pi(\theta, \mathcal{D}) = -\sum_t \log \pi_\theta(a_t|s_t)(G_t - v_\theta(s_t)) \tag{4}$$

$$\mathcal{L}_v(\theta, \mathcal{D}) = \frac{1}{2}\sum_t (G_t - v_\theta(s_t))^2 \tag{5}$$

$$\mathcal{L}_{\text{ent}}(\theta, \mathcal{D}) = -\sum_t \sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t), \tag{6}$$

where $\alpha_\pi, \alpha_v, \alpha_{\text{ent}}$ are the weights of individual losses, t are the environment step indices of transition tuples (s_t, r_t, a_t, s_{t+1}) in $\mathcal{D}$, and G_t is a return estimate at timestep t. The hyperparameters required for computation of each of the three terms (for example, discounts γ and bootstrapping parameters λ in n-step returns Sutton & Barto (2018)) or the weights of individual losses, can become tunable meta-parameters.

The outer loss objective generally consists of the same three terms, weighted by hyperparameters $\alpha_\pi^{(\text{outer})}, \alpha_v^{(\text{outer})}, \alpha_{ent}^{(\text{outer})}$:

$$\mathcal{L}^{(\text{outer})}(\eta, \mathcal{D}) = \alpha_\pi^{(\text{outer})} \mathcal{L}_\pi(\theta(\eta), \mathcal{D}) + \alpha_v^{(\text{outer})} \mathcal{L}_v(\theta(\eta), \mathcal{D}) + \alpha_{\text{ent}}^{(\text{outer})} \mathcal{L}_{\text{ent}}(\theta(\eta), \mathcal{D}), \tag{7}$$

where the data $\mathcal{D}$ used for estimating the outer loss, could be coming from a separate rollout or the same data as used in the inner loss.

A.3 Bootstrapped Meta-Gradients: $Q(\lambda)$ agent

An alternative outer loss is proposed in Flennerhag et al. (2022): the meta-parameters are trained to minimize a distance to a target which has been bootstrapped from the meta-learner. This method is referred to as Bootstrapped Meta-Gradients (BMG). In the best studied version of BMG, the meta-gradients are propagated through the last K updates, while minimizing the KL-divergence between the policy parametrized with θ_K and a bootstrap target $\pi_{\hat{\theta}}$ obtained by optimizing the policy for another $L-1$ steps under the meta-learned update rule:

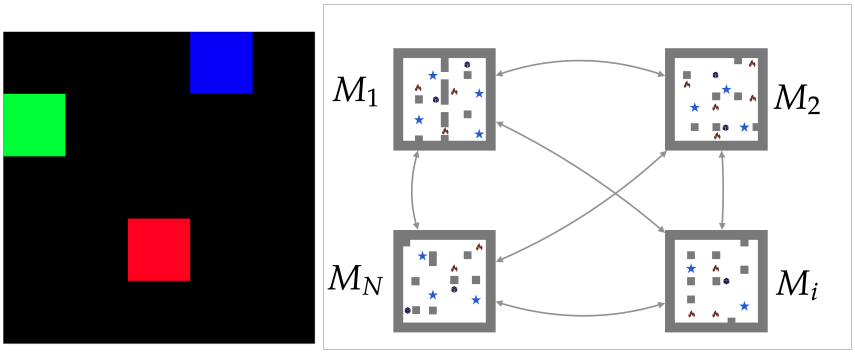
$$\mathcal{L}_{\text{BMG}}^{(\text{outer})} = KL(\pi_{\hat{\theta}} || \pi_{\theta_K}), \tag{8}$$

where the hyperparameter L is referred to as bootstrap target length. The use of target which is $L-1$ steps ahead (without increasing the number of steps the meta-gradients are backpropagating through) reduces the myopia of standard MG objectives, while the use of KL divergence reduces ill-conditioning of outer loop objective. The resulting adaptations of meta-parameters encourage reaching the target policy in a smaller number of inner optimization steps.

When tuning the exploration rate ϵ of $Q(\lambda)$ agents, we use the following implementation from Flennerhag et al. (2022) to make the outer objective differentiable with respect to ϵ. In equation 8, the stochastic bootstrap policy $\pi_{\theta_K}(a|s)$ and the target policy $\pi_{\hat{\theta}}(a|s)$ are defined as:

$$\pi_{\theta_K}(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if a=}\arg\max_{a'} \mathrm{q}_{\theta_K}(s, a') \\ \frac{\epsilon_k}{|A|} & \text{else.} \end{cases} \qquad \pi_{\hat{\theta}}(a|s) = \begin{cases} 1 & \text{if a=}\arg\max_{a'} \mathrm{q}_{\hat{\theta}}(s, a') \\ 0 & \text{else.} \end{cases} \tag{9}$$

, where $q_{\theta_K}(s, a)$ is the learned value function, the parameters $\hat{\theta}$ are once again obtained by taking another $L-1$ update steps, and $|A|$ is the number of different actions. The resulting objective does



(a) Two Colors. **(b)** Switching MDPs.

Figure 6: Illustration of Two Colors and Switching MDPs environments. In Two Colors, the agent (green) is tasked with navigating to either blue or red square. In Switching MDPs, the reward and transition function changes to one of the N predefined randomly generated options at regular intervals.

not require differentiation through the update-rule, hence as in Flennerhag et al. (2022), we use $K = 0$.

A.4 Environments

In this section, we provide additional information about the environments used in this paper. The Two Colors environment is visualised in Figure 6a and Switching MDPs in 6b.

A.4.1 Two Colors

The dimension of the grid in Two Colors environment is 5×5. The observation space is constructed by concatenating one-hot encodings of x and y coordinates of positions of agents and two other objects. The total dimension of resulting observation space is $3 \times 2 \times 5 = 30$.

A.4.2 Switching MDPs

The reward function for each MDP is generated in the following manner: for each state-action pair there is a 50% chance of zero reward, 20% chance of reward of +1, 20% chance of reward -1, and 10% chance of a random reward sampled uniformly from the interval [-1, 1]. The transition function for each MDP is a standard two dimensional grid world—states are characterized by an (x, y) coordinate, and there are four actions that move the agent up, left, right, and down unless the intended cell is the edge of the grid or a wall is present. The transition function changes across the N MDPs by the addition of a set of walls randomly placed throughout the grid. Up to 15 walls are placed per-MDP, in sequence, by randomly sampling unoccupied cells (by goal, agent, or another wall).

At the beginning of an experiment, we generate a set of N different MDPs. At regular fixed intervals, the next grid world is sampled from this set uniformly with replacement. The set and the order of samples is uniquely determined by experiment random seed. If N is large compared to the total number of task switches experienced during a lifetime (e.g. $N = 1000$ with a change period $100,000$ and a lifetime of $20M$ steps), the probability of agent experiencing the same task multiple times is small. Note that since we measure the lifetime performance after $20M$ environment steps in our experiments, N does not correspond to the number of different MDPs experienced during this lifetime.

A.5 Experimental Setup and Hyper-parameters

In this section, we provide further details on the experimental setup and hyper-parameters of agents and meta-learners used in Section 6.

The hyper-parameters used in the experiments with AC agents are described in Table 3. Our implementation of the AC agent implements the softmax policy and the value function as two separate

Table 3: Hyper-parameters used in experiments with Actor-Critic agents. We denote in italic when a hyper-parameter is required in only some variants of the experiments (e.g. AC baseline, MG or BMG objective, contextual meta-gradients).

Inner Learner: Actor Critic	
Optimizer	SGD
Learning Rate	0.1
Batch Size	16
α_{ent} candidates *(AC only)*	$[0, 0.1, 0.2, 0.4, 0.8]$
γ	0.99
MLP hidden layers (v, π)	2
MLP feature size (v, π)	256
Activation Function	ReLU

Meta-learner	
Optimizer	Adam
ϵ (Adam)	10^{-4}
β_1, β_2 (Adam)	0.9, 0.999
Learning Rate candidates	$[10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}]$
$\alpha_{\text{ent}}^{(\text{outer})}$ *(MG only)*	$[0, 0.1]$
K candidates	$[1, 3, 6]$
L candidates *(BMG only)*	$[8, 16]$
H *(contextual only)*	10
MLP hidden layers *(contextual only)*	2
MLP feature size *(contextual only)*)	64
Activation Function *(contextual only)*	ReLU
Output Activation	Sigmoid

Table 4: Hyper-parameters used in experiments with $Q(\lambda)$ agents. We denote in italic when a hyper-parameter is required in only some variants of the experiments (e.g. $Q(\lambda)$ baseline, contextual meta-gradients).

Inner Learner: $Q(\lambda)$	
Optimizer	Adam
ϵ (Adam)	10^{-4}
β_1, β_2 (Adam)	0.9, 0.999
Learning Rate candidates *($Q(\lambda)$ only)*	$[3 \cdot 10^{-3}, 10^{-4}, 3 \cdot 10^{-5}, 10^{-5}]$
ϵ candidates *($Q(\lambda)$ only)*	$[0.3, 0.1, 0.03, 0.01]$
λ	0.9
γ	0.99
MLP hidden layers (q)	2
MLP feature size (q)	256
Activation Function	ReLU

Meta-learner	
Optimizer	Adam
ϵ (Adam)	10^{-4}
β_1, β_2 (Adam)	0.9, 0.999
Learning Rate candidates *(no context)*	$[10^{-2}, 3 \cdot 10^{-3}, 10^{-3}, 3 \cdot 10^{-4}]$
Learning Rate candidates *(with context)*	$[10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}]$
L candidates	$[16, 32, 128]$
H *(contextual only)*	100
MLP hidden layers *(contextual only)*	2
MLP feature size *(contextual only)*)	128
Activation Function *(contextual only)*	ReLU
Output Activation	Sigmoid

feed-forward MLPs. The parameters are updated every 16 environment steps: given fixed parameters, the agent interacts with the environment for 16 steps collecting observations, rewards and

Table 5: Summary of all context features used in experiments with AC agents. When description of $f_i^{(feat)}$ includes mean & std, we are indicating that $f_i^{(feat)}$ is mean and standard deviation of the specified quantity, over the rollout data used to compute i-th update. Here we denoted the gradient with respect to inner loss at i-th update with $\nabla_\theta \mathcal{L}_i$.

Feature Type	$f_i^{(feat)}$	Feature Dimension
Reward	r_t (mean & std)	$2 \times H$
Value	$v(s_t; \theta_i)$ (mean & std)	$2 \times H$
TD-Error	$r_t + \gamma v(s_{t+1}; \theta_i) - v(s_t; \theta_i)$ (mean & std)	$2 \times H$
Action Probabilities	$\pi(a\|s_t; \theta_i)$ (mean & std)	$8 \times H$
States	s_t (mean & std)	$60 \times H$
Cosine Distance Between Gradients	$1 - \frac{\nabla_\theta \mathcal{L}_{i-1} \nabla_\theta \mathcal{L}_{i-2}}{\|\nabla_\theta \mathcal{L}_{i-1}\| \|\nabla_\theta \mathcal{L}_{i-2}\|}$	H
Meta-Parameters	η_{i-1}	H

Table 6: Summary of all context features used in experiments with $Q(\lambda)$ agents.

Feature Type	$f_i^{(feat)}$	Feature Dimension
Reward	r_t	H
Value	$q(a_t, s_t; \theta_i)$	H
TD-Error	$r_t + \gamma \max_a q(s_{t+1}, a; \theta_i) - q(s_t, a_t; \theta_i)$	H

actions into a rollout $\mathcal{D}$, which is then used to compute the inner loss. The inner loss consists of the following four terms (policy, value and entropy and L_2 loss respectively):

$$\mathcal{L}^{(\text{inner})}(\theta, \mathcal{D}) = \mathcal{L}_\pi(\theta, \mathcal{D}) + \mathcal{L}_v(\theta, \mathcal{D}) + \alpha_{\text{ent}} \mathcal{L}_{\text{ent}}(\theta, \mathcal{D}) + \alpha_{\text{L2}} \mathcal{L}_{\text{L2}}(\theta) \tag{10}$$

In most of the experiments, the only meta-parameter is α_{ent}, and $\alpha_{L2} = 0$ (i.e. there is no L_2 regularization). However, in Section 6.3 (Figure 3b), we tune both α_{ent} and α_{L2}. There we used two separate MLPs to predict each of the two meta-parameters. The output of the meta-parameter network predicting α_{L2} has been scaled by a fixed quantity (10^{-4}) to prevent instabilities caused by too strong forgetting. The weights of both policy and value networks were included in $\mathcal{L}_{\text{L2}}(\theta)$.

The hyper-parameters used in the experiments with $Q(\lambda)$ agents are described in Table 6. The q-function is again a feed-forward MLP. The agent is optimized at each step, i.e. without batching. To avoid instabilities this could cause, we use a momentum term that maintains an exponentially moving average over gradients, with the discount factor 0.9. We sweep over the learning rate of the inner learner only for the fixed meta-parameter baseline, for the meta-gradient methods, the inner learner's learning rate is set to $3 \cdot 10^{-5}$.

To ensure stable initial predictions, the meta-networks were pre-trained on random context inputs sampled from uniform distribution $[-1, 1]$, to predict an output in the middle of the possible meta-parameter range (0.5 for α_{ent} and ϵ, $5 \cdot 10^{-5}$ for α_{L2}). Note that when using contextual meta-gradients, we do not sweep over the hyper-parameters introduced by the addition of context (e.g. the size of meta-network and context history H).

A.6 Context Features

In this section, we describe in more details the construction of context features. At each update step i, the meta-parameters are computed by feeding context features c_i into a meta-parameter function. The context features are a concatenation of one or several different types of features (e.g. reward, value, TD-error), the type of features used in each experiments is described in the corresponding experiment's section. For example, when the context is specified as *Reward*, the input to the meta-parameter function is $c_i = [c_i^{(reward)}]$, and when the context is *Rich*, the input to the meta-parameter function is $c_i = [c_i^{(reward)}, c_i^{(value)}, c_i^{(td-error)}]$. For each different feature type (here denoted with "feat"), the features are a history of size H:

$$c_i^{(feat)} = [\hat{f}_i^{(feat)}, \hat{f}_{i-1}^{(feat)}, ..., \hat{f}_{i-H+1}^{(feat)}], \tag{11}$$

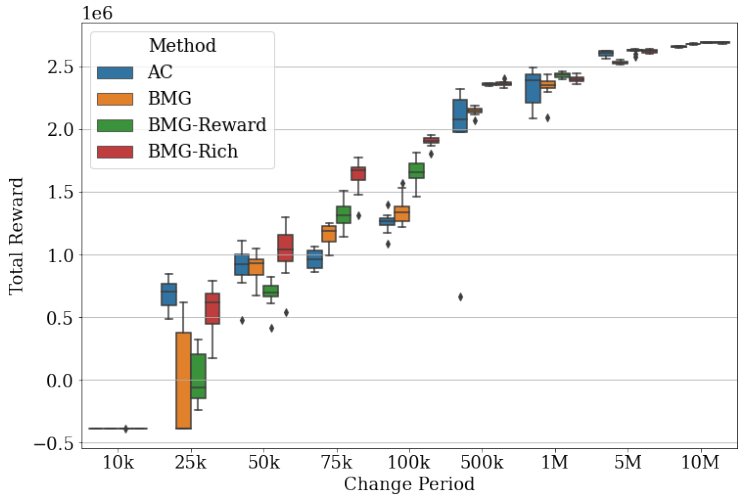


Figure 7: Two Colors: Comparison of methods under different rates of environment non-stationarity as measured in total return after 10M steps. The medians and interquartile ranges are computed over 10 random seeds. The regime in which meta-gradients provide a significant advantage over the baseline is the greatest for meta-gradients with rich context (50k-500k steps between task switches). All meta-gradient fail to beat the baseline when the rate of non-stationarity is too high (25k steps between task switches).

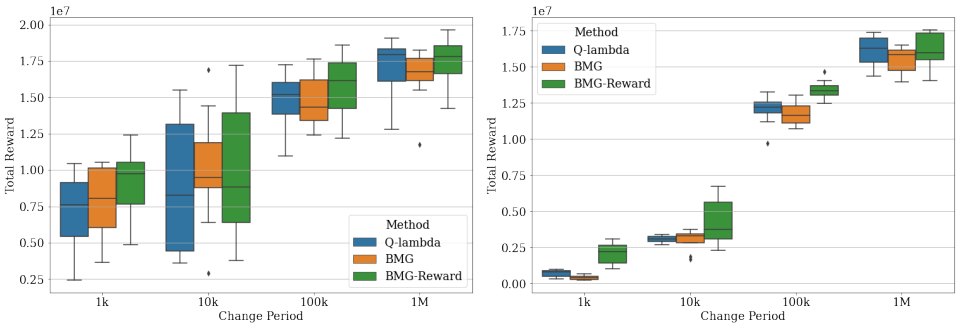
where $\hat{f}_i^{(feat)}$ is a quantity computed using statistics at i-th update step. Each of these quantities has been normalized (we used the statistics of context features measured during the first 2M steps of training with the AC baseline agent, but more generally, one could use a running average as an estimate) and passed through a $tanh(\cdot)$ function to ensure all features are in the [-1, 1] range. We will refer with $f_i^{(feat)}$ to quantities computed before this transformation. For example, in experiments described with AC-BMG-Reward, each $f_i^{(reward)}$ is a mean of rewards r_t from the rollout D_i which was used to compute i-th update. In experiments described with $Q(\lambda)$-BMG-Reward, because i-th update for $Q(\lambda)$ agents is calculated using data from only one environment step, $f_i^{(reward)}$ is the reward r_t from only that step.

The summary of all different feature types used in experiments with AC agents, including how $f_i^{(feat)}$ is defined for that feature type and dimensions of that feature, can be be found in Table 5. Note though that when the feature is specified as *Reward* (AC-MG-Reward and AC-BMG-Reward), we only used mean of rewards, and the corresponding feature dimension is H. The summary of all different feature types used in experiments with $Q(\lambda)$ agents can be found in Table 6.

A.7 Different Rates of Non-Stationarity: Total Rewards

Lastly, we supplement the results in Section 6.4 by illustrating how the performance of all methods drops as the rate of environment non-stationarity increases.

In Two Colors experiments (Figure 7), no learning occurs when the environment switches every 10k steps. In Switching MDPs, note that the performance drop is much more significant when the number of different MDPs is large (Figure 8b), indicating that in this environment, the agents benefit from repeated exposure to the same MDP.



(a) 4 MDPs: Comparison of $Q(\lambda)$, $Q(\lambda)$-BMG and $Q(\lambda)$-BTMG-Reward.

(b) 1000 MDPs: Comparison of comparison of $Q(\lambda)$, $Q(\lambda)$-BMG and $Q(\lambda)$-BTMG-Reward.

Figure 8: Comparison of methods (measured in total return after 10M steps) under different rates of environment non-stationarity: (a) Switching MDPs Environment with 4 different MDPs, (b) Switching MDPs Environment with 1000 different MDPs. The performance of all methods drops when the number of different tasks is very large and as the rate of non-stationarity increases. Meta-gradients with Reward context perform the best when the rate of non-stationarity is high and the number of different tasks are high.