arXiv:2109.04504v1 [cs.LG] 9 Sep 2021

Bootstrapped Meta-Learning

Sebastian Flennerhag
DeepMind
flennerhag@google.com

Yannick Schroecker
DeepMind

Tom Zahavy
DeepMind

Hado van Hasselt
DeepMind

David Silver
DeepMind

Satinder Singh
DeepMind

Abstract

Meta-learning empowers artificial intelligence to increase its efficiency by learning how to learn. Unlocking this potential involves overcoming a challenging meta-optimisation problem that often exhibits ill-conditioning, and myopic meta-objectives. We propose an algorithm that tackles these issues by letting the meta-learner teach itself. The algorithm first bootstraps a *target* from the meta-learner, then optimises the meta-learner by minimising the *distance* to that target under a chosen (pseudo-)metric. Focusing on meta-learning with gradients, we establish conditions that guarantee performance improvements and show that the improvement is related to the target distance. Thus, by controlling curvature, the distance measure can be used to ease meta-optimization, for instance by reducing ill-conditioning. Further, the bootstrapping mechanism can extend the effective meta-learning horizon without requiring backpropagation through all updates. The algorithm is versatile and easy to implement. We achieve a new state-of-the art for model-free agents on the Atari ALE benchmark, improve upon MAML in few-shot learning, and demonstrate how our approach opens up new possibilities by meta-learning efficient exploration in an ε-greedy Q-learning agent.

1 Introduction

In a standard machine learning problem, a *learner* or *agent* learns a task by iteratively adjusting its parameters under a given update rule, such as Stochastic Gradient Descent (SGD). Typically, the learner's update rule must be tuned manually for each task. In contrast, humans learn new tasks seamlessly by relying on previous experiences to inform their learning processes [53].

For a (machine) learner to have the same capability, it must be able to learn its update rule (or such inductive biases). *Meta-learning* is one approach that learns (parts of) an update rule by applying it for some number of steps and then evaluating the resulting performance [47, 24, 10]. For instance, a well-studied and often successful approach is to tune parameters of a gradient-based update, either online during training on a single task [9, 34, 66, 68], or meta-learned over a distribution of tasks [16, 46, 18, 26, 13]. More generally, the update rule can be an arbitrary parameterised function [25, 4, 39], or the function itself can be meta-learned jointly with its parameters [2, 44].

Meta-learning is challenging because to evaluate an update rule, it must first be applied. This often leads to high computational costs. As a result most works optimise performance after K applications of the update rule and assume that this yields improved performance for the remainder of the learner's lifetime [10, 34, 35]. When this assumption fails, meta-learning suffers from a short-horizon bias [65, 35]. Similarly, optimizing the learner's performance after K updates can fail to account for the *process* of learning, causing another form of myopia [17, 54, 12, 11]. Challenges in meta-optimisation have been observed to cause degraded lifetime performance [32, 63], collapsed exploration [54, 12], biased learner updates [54, 69], and poor generalisation performance [65, 67, 59].

We argue that defining the meta-learner's objective directly in terms of the learner's objective—i.e. the performance after K update steps—creates two bottlenecks in meta-optimisation. The first bottleneck is curvature: the meta-objective is constrained to the same type of geometry as the learner; the second is myopia: the meta-objective is fundamentally limited to evaluating performance within the K-step horizon, but ignores future learning dynamics. Our goal is to design an algorithm that removes these.

The algorithm relies on two main ideas. First, to mitigate myopia, we introduce the notion of bootstrapping a target from the meta-learner itself or from some other update rule, what we refer to as a *meta-bootstrap*. This allows us to infuse information about learning dynamics into the objective. Second, to control curvature, we formulate the meta-objective in terms of minimising distance (or divergence) to the bootstrapped target. This gives us control over the optimisation landscape. The key insight is that the meta-learner can learn to learn from itself by trying to match future desired updates with fewer steps. This leads to a bootstrapping effect, where improvements beget improvements.

We present a detailed formulation in Section 3; on a high level, as in previous works, we first unroll the meta-learned update rule for K steps to obtain the learner's new parameters. Whereas standard meta-objectives optimise the update rule with respect to (w.r.t.) the learner's performance under the new parameters, our proposed algorithm constructs the meta-objective in two steps:

1. It *bootstraps* a *target* from the learner's new parameters. In this paper, we generate targets by continuing to update the learner's parameters—either under the meta-learned update rule or another update rule—for some number of steps.
2. The learner's new parameters—which are a function of the meta-learner's parameters—and the target are projected onto a *matching space*. A simple example is Euclidean parameter space. To control curvature, we may choose a different (pseudo-)metric space. For instance, a common choice under probabilistic models is the Kullback-Leibler (KL) divergence.

The meta-learner is optimised by minimising distance to the bootstrapped target. We focus on gradient-based optimisation, but other optimisation routines are equally applicable. By optimising meta-parameters in a well-behaved space, we can drastically reduce ill-conditioning and other phenomena that disrupt meta-optimisation. In particular, this form of *Bootstrapped Meta-Gradient* (BMG) enables us to infuse information about *future* learning dynamics without increasing the number of update steps to backpropagate through. In effect, the meta-learner becomes its own teacher. Because BMG optimises the meta-learner to reach a desired future target state faster, it embodies a form of acceleration akin to Nesterov Momentum [37] and Raphson-Newton updates. We show that BMG can guarantee performance improvements (Theorem 1) and that this guarantee can be stronger than under standard meta-gradients (Corollary 1). Empirically, we find that BMG provides substantial performance improvements over standard meta-gradients in various settings. We obtain a new state-of-the-art result for model-free agents on Atari (Section 5.2) and improve upon MAML [16] in the few-shot setting (Section 6). Finally, we demonstrate how BMG enables new forms of meta-learning, exemplified by meta-learning ε-greedy exploration (Section 5.1).

2 Related Work

The idea of bootstrapping, as used here, stems from temporal difference (TD) algorithms in reinforcement learning (RL) [55]. In these algorithms, an agent learns a value function by using its own future predictions as targets. Bootstrapping has recently been introduced in the self-supervised setting [21, 20]. In this paper, we introduce the idea of bootstrapping in the context of meta-learning, where a meta-learner learns about an update rule by generating future targets from it.

Our approach to target matching is related to methods in multi-task meta-learning [17, 38] that meta-learn an initialisation for SGD by minimising the Euclidean distance to task-optimal parameters. BMG generalise this concept by allowing for arbitrary meta-parameters, matching functions, and target bootstraps. It is further related the more general concept of self-referential meta-learning [47, 48], where the meta-learned update rule is used to optimise its own meta-objective.

Target matching under KL divergences results in a form of distillation [23], where an online network (student) is encouraged to match a target network (teacher). In a typical setup, the target is either a fixed (set of) expert(s) [23, 45] or a moving aggregation of current experts [57, 20], whereas BMG bootstraps a target by following an update rule. Finally, BMG is loosely inspired by trust-region methods that introduce a distance function to regularize gradient updates [41, 51, 58, 22].

3 Bootstrapped Meta-Gradients

We begin in the single-task setting and turn to multi-task meta-learning in Section 6. The learner's problem is to minimize a stochastic objective $f(\mathbf{x}) := \mathbb{E}[\ell(\mathbf{x}; \boldsymbol{\zeta})]$ over a data distribution $p(\boldsymbol{\zeta})$, where $\boldsymbol{\zeta}$ denotes a source of data and $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^{n_x}$ denotes the learner's parameters. In RL, f is typically the expected value of a policy $\pi_{\mathbf{x}}$; in supervised learning, f may be the expected negative log-likelihood under a probabilistic model $\pi_{\mathbf{x}}$. We provide precise formulations in Section 5 and Section 6, respectively.

The meta-learner's problem is to learn an update rule $\varphi : \mathcal{X} \times \mathcal{H} \times \mathcal{W} \to \mathcal{X}$ that updates the learner's parameters by $\mathbf{x}^{(1)} = \mathbf{x} + \varphi(\mathbf{x}, \mathbf{h}, \mathbf{w})$ given $\mathbf{x} \in \mathcal{X}$, a learning state $\mathbf{h} \in \mathcal{H}$, and meta-parameters $\mathbf{w} \in \mathcal{W} \subset \mathbb{R}^{n_w}$ of the update rule. We make no assumptions on the update rule other than differentiability in $\mathbf{w}$. As such, φ can be a recurrent neural network [25, 62, 4] or gradient descent [9, 34, 16]. The learning state $\mathbf{h}$ contains any other data required to compute the update; in a black-box setting $\mathbf{h}$ contains an observation and the recurrent state of the network; for gradient-based updates, $\mathbf{h}$ contains the (estimated) gradient of f at $\mathbf{x}$ along with any auxiliary information, such as momentum vectors. For instance, SGD is given by the update $\mathbf{x}^{(1)} = \mathbf{x} - \alpha \nabla_x f(\mathbf{x})$ with $\mathbf{h} = \nabla_x f(\mathbf{x})$, $\mathbf{w} = \alpha \in \mathbb{R}_+$.

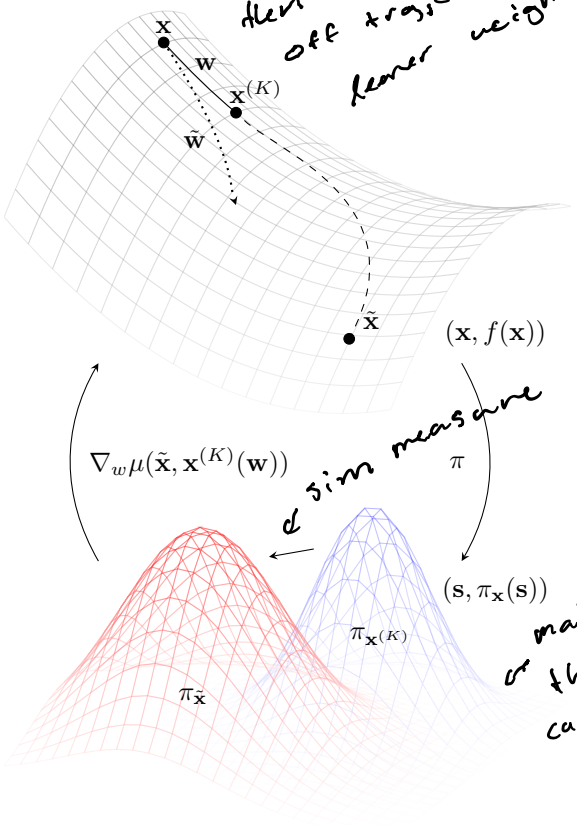

Figure 1: Bootstrapped Meta-Gradients.

The standard meta-gradient (MG) optimises meta-parameters $\mathbf{w}$ by taking K steps under φ and evaluating the resulting learner parameter vector under f. With a slight abuse of notation, let $\mathbf{x}^{(K)}(\mathbf{w})$ denote the learner's parameters after K applications of φ starting from some $(\mathbf{x}, \mathbf{h}, \mathbf{w})$, where $(\mathbf{x}, \mathbf{h})$ evolve according to φ and the underlying data distribution. The MG update is defined by

$$\mathbf{w}' = \mathbf{w} - \beta \nabla_{\mathbf{w}} f\left(\mathbf{x}^{(K)}(\mathbf{w})\right), \qquad \beta \in \mathbb{R}_+ . \tag{1}$$

Common extensions involve averaging the performance over all iterates $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(K)}$ [4, 12, 5] or using validation data in the meta-objective [10, 34, 16, 66]. We observe two bottlenecks in the meta-objective in Eq. 1. First, the meta-objective is subject to the same curvature as the learner. Thus if f is ill-conditioned, so will the meta-objective be. Second, the meta-objective is only able to evaluate the meta-learner on dynamics up to the Kth step, but ignores effects of future updates.

To tackle myopia, we introduce a *Target Bootstrap* (TB) $\xi : \mathcal{X} \mapsto \mathcal{X}$ that maps the meta-learner's output $\mathbf{x}^{(K)}$ into a bootstrapped target $\tilde{\mathbf{x}} = \xi(\mathbf{x}^{(K)})$. We focus on TBs that unroll φ a further $L-1$ steps before taking final gradient step on f, producing targets of the form $\tilde{\mathbf{x}} = \mathbf{x}^{K+L-1} - \nabla f(\mathbf{x}^{K+L-1})$. Intuitively, this TB encourages the meta-learner to reach future states on its trajectory faster while nudging the trajectory in a descent direction. Crucially, regardless of the bootstrapping strategy, we do *not* backpropagate through the target. Akin to temporal difference learning in RL [55], the target is a fixed goal that the meta-learner should try to produce within the K-step budget.

Finally, to improve the meta-optimisation landscape, we introduce a *matching function* $\mu : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ that measures the (dis)similarity between the meta-learner's output, $\mathbf{x}^{(K)}(\mathbf{w})$, and the target, $\tilde{\mathbf{x}}$, in a matching space defined by μ (see Figure 1). Taken together, the BMG update is defined by

$$\tilde{\mathbf{w}} = \mathbf{w} - \beta \nabla_{\mathbf{w}} \mu\left(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}(\mathbf{w})\right), \qquad \beta \in \mathbb{R}_+, \tag{2}$$

where the gradient is with respect to the second argument of μ. Thus, BMG describes a family of algorithms based on the choice of matching function μ and TB ξ. In particular, MG is a special case of BMG under matching function $\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) = \|\tilde{\mathbf{x}} - \mathbf{x}^{(K)}\|_2^2$ and TB $\xi(\mathbf{x}^{(K)}) = \mathbf{x}^{(K)} - \frac{1}{2}\nabla_x f(\mathbf{x}^{(K)})$, since the bootstrapped meta-gradient reduces to the standard meta-gradient:

$$\nabla_w \left\| \tilde{\mathbf{x}} - \mathbf{x}^{(K)}(\mathbf{w}) \right\|_2^2 = -2D\left(\tilde{\mathbf{x}} - \mathbf{x}^{(K)}\right) = D\nabla_x f\left(\mathbf{x}^{(K)}\right) = \nabla_w f\left(\mathbf{x}^{(K)}(\mathbf{w})\right), \tag{3}$$

where D denotes the (transposed) Jacobian of $\mathbf{x}^{(K)}(\mathbf{w})$. For other matching functions and target strategies, BMG produces different meta-updates compared to MG. We discuss these choices below.

Matching Function Of primary concern to us are deep neural networks that output a probabilistic distribution, $\pi_{\mathbf{x}}$. A common pseudo-metric over a space of probability distributions is the Kullback-Leibler (KL) divergence. For instance, Natural Gradients [3] point in the direction of steepest descent under the KL-divergence. In neural networks, this is often approximated through a KL-regularization term [41], which is an instance of Mirror Descent [7]. KL-divergences also arise naturally in RL algorithms [28], often in the form of trust-region methods [51, 52, 1]. Hence, a natural starting point is to consider KL-divergences between the target and the iterate, e.g. $\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) = \mathrm{KL}\left(\pi_{\tilde{\mathbf{x}}} \parallel \pi_{\mathbf{x}^{(K)}}\right)$. In actor-critic algorithms [56], the policy defines only part of the agent—the value function defines the other. Thus, we also consider a composite matching function over both policy and value function.

Target Bootstrap We analyze conditions under which BMG guarantees performance improvements in Section 4 and find that the target should co-align with the gradient direction. Thus, in this paper we focus on gradient-based TBs and find that they perform well empirically. As with matching functions, this is a small subset of all possible choices; we leave the exploration of other choices for future work.

4 Performance Guarantees

In this analysis, we restrict attention to the noise-less setting (true expectations) and consider 1-step target updates. In this setting, we ask three questions: (1) what local performance guarantees are provided by MG? (2) What performance guarantees can BMG provide? (3) How do these guarantees relate to each other? To answer these questions, we analyse how the performance around $f(\mathbf{x}^{(K)}(\mathbf{w}))$ changes by updating $\mathbf{w}$ either under standard meta-gradients (Eq. 1) or bootstrapped meta-gradients (Eq. 2). Throughout, we assume that f and $\mathbf{x}^{(K)}$ are Lipschitz.

First, consider improvements under the MG update. In online optimisation, the MG update can achieve strong convergence guarantees if the problem is well-behaved [60], with similar guarantees in the multi-task setting [6, 30, 13]. A central component of these results is that the MG update guarantees a local improvement in the objective. Lemma 1 below presents this result in our setting, with the following notation: let $\|\mathbf{u}\|_A := \sqrt{\langle \mathbf{u}, A\,\mathbf{u}\rangle}$ for any square real matrix A. Let $G^T = D^T D \in \mathbb{R}^{n_x \times n_x}$, with $D := \left[\frac{\partial}{\partial \mathbf{w}} \mathbf{x}^{(K)}(\mathbf{w})\right]^T \in \mathbb{R}^{n_w \times n_x}$. Note that $\nabla_w f(\mathbf{x}^{(K)}(\mathbf{w})) = D\nabla_x f(\mathbf{x}^{(K)})$.

Lemma 1 (MG Descent)**.** *Let $\mathbf{w}'$ be given by Eq. 1. For β sufficiently small, $f\left(\mathbf{x}^{(K)}(\mathbf{w}')\right) - f\left(\mathbf{x}^{(K)}(\mathbf{w})\right) = -\beta\|\nabla_x f(\mathbf{x}^{(K)})\|_{G^T}^2 + O(\beta^2) < 0$.*

We defer all proofs to Appendix A. Lemma 1 relates the gains obtained under standard meta-gradients to the local gradient norm of the objective. A key property of the meta-gradient is that, because the meta-objective is given by f, it is not scale-free [c.f. 50], nor invariant to re-parameterisation. Thus, if f is a highly non-linear function, the meta-gradient can vary widely, preventing efficient performance improvement. Next, we turn to BMG. To facilitate the analysis we assume μ differentiable, Lipschitz continuous and convex in its second argument, with 0 being its minimum:

$$\mu(\mathbf{x}, \mathbf{z}) = 0 \iff \mathbf{x} = \mathbf{z}, \qquad \mu(\mathbf{x}, \mathbf{z}) \geq 0 \qquad \forall\, \mathbf{x}, \mathbf{z} \in \mathcal{X}\,. \tag{4}$$

To obtain performance improvement guarantees, we must place some structure on the TB. This is because if $\tilde{\mathbf{x}}$ points away from the solution space, updating the meta-learner in this direction will not improve the update rule. For instance, this can happen if φ is a black-box meta-learner and the target is generated by unrolling φ. To establish a descent guarantee, we first analyse a gradient-based TB of the form $\xi_G^\alpha(\mathbf{x}^{(K)}) := \mathbf{x}^{(K)} - \alpha G^T \nabla_x f(\mathbf{x}^{(K)}), \alpha \in \mathbb{R}_+$, which provides an intuitive guarantee.

Theorem 1 (BMG Descent)**.** *Let* $\tilde{\mathbf{w}}$ *be given by Eq. 2 with target* $\tilde{\mathbf{x}} = \xi_G^\alpha(\mathbf{x}^{(K)})$*. For* α, β *sufficiently small,* $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) = -\frac{\beta}{\alpha}\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) + O(\beta(\alpha+\beta)) < 0.$

Theorem 1 establishes that BMG under ξ_G^α yields performance improvements that are proportional to the meta-loss itself, $-\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)})$. Thus, the matching function controls the dynamics of the meta-optimisation problem. While BMG is not fully invariant to re-parameterisation (as the target relies on the gradient of f), it can guarantee larger improvements to the update rule than MG—in particular, under the (squared) Euclidean norm. To show this, let $r \coloneqq \|\nabla f\left(\mathbf{x}^{(K)}\right)\|_2 / \|G^T \nabla f\left(\mathbf{x}^{(K)}\right)\|_2$ denote the gradient norm ratio.

Corollary 1. *Let* $\mu = \|\cdot\|_2^2$ *and* $\tilde{\mathbf{x}} = \xi_G^r(\mathbf{x}^{(K)})$*. Let* $\mathbf{w}'$ *be given by Eq. 1 and* $\tilde{\mathbf{w}}$ *be given by Eq. 2. For* β *sufficiently small,* $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) \leq f\big(\mathbf{x}^{(K)}(\mathbf{w}')\big)$*, strictly if* $GG^T \neq G^T$ *and* $G^T \nabla_x f(\mathbf{x}^{(K)}) = \mathbf{0}$*.*

While these results provide theoretical support for BMG, they rely on an impractical target bootstrap that requires computing a projection matrix $G \in \mathbb{R}^{n_x \times n_x}$. For this reason, it is of interest to relax the assumptions on the target. In general, relaxing restriction on the target requires tightening them on the matching function. For instance, if μ is the Euclidean norm, Eq. 3 implies that the TB $\xi(\mathbf{x}^{(K)}) = \mathbf{x}^{(K)} - \frac{1}{2}\nabla_x f(\mathbf{x}^{(K)})$ guarantees an improvement by virtue of Lemma 1. To extend Theorem 1, we establish sufficient (but not necessary) conditions that guarantee an improvement.

Corollary 2. *For any* ξ*, let* $\tilde{\mathbf{w}}$ *be given by Eq. 2. Define* $\bar{\mathbf{x}} = \xi_G^\beta(\mathbf{x}^{(K)})$*. For* β *sufficiently small, if* $\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) > \mu(\tilde{\mathbf{x}}, \bar{\mathbf{x}})$*, then* $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) < 0.$

The gist of Corollary 2 is that any target that produces updates in the neighborhood of the "ideal" update in Theorem 1 guarantees an improvement. This condition is not sufficient because the target can be on the same tangent but further away–so that the condition in Corollary 2 fails—but would clearly guarantee an improvement by reducing β accordingly. The implication of Corollary 2 is that bootstraps that align with $\nabla f(\mathbf{x}^{(K)})$ are likely to yield performance improvements. This is because the bootstrap in Theorem 1, $G\nabla f(\mathbf{x}^{(K)}$, projects the gradient by a positive semi-definite matrix. An upshot of this is that strong curvature in the update rule (manifest by G) can cause the gradient of f to be a poor guide for the target bootstrap.

Discussion Our analysis focuses on an arbitrary (albeit noiseless) objective f and establishes that BMG can guarantee improved performance under a relatively simple TB. We further show that BMG can yield larger local improvements than MG. While the exact form of the TB in Theorem 1 is impractical, Corollary 2 provides some insights into what kind of targets are permissible. Empirically, we find that standard (non-preconditioned) steps perform well, implying that the projection G^T is not essential in practice. Our analysis do not address the question of whether using a target that is "far" away guarantees an improvement. This need not be the case in the non-convex setting. Empirically, we observe clear benefits from bootstraps that unroll the meta-learner for several steps before taking a gradient step on f. Exploring other forms of bootstraps is an exciting area for future research.

5 Reinforcement Learning

We consider a typical reinforcement learning problem, modelled as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. Given an initial state $\mathbf{s}_0 \in \mathcal{S}$, at each time step $t \in \mathbb{N}$, the agent takes an action $\mathbf{a}_t \sim \pi_{\mathbf{x}}(\mathbf{a} \mid \mathbf{s}_t)$ from a *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ parameterised by $\mathbf{x}$. The agent obtains a reward $r_{t+1} \sim \mathcal{R}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ based on the transition $\mathbf{s}_{t+1} \sim \mathcal{P}(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$. The *action-value* of the agent's policy given a state $\mathbf{s}_0$ and action $\mathbf{a}_0$ is given by $Q_{\mathbf{x}}(\mathbf{s}_0, \mathbf{a}_0) \coloneqq \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid \mathbf{s}_0, \mathbf{a}_0, \pi_{\mathbf{x}}]$ under discount rate $\gamma \in [0, 1)$. The corresponding *value* of policy $\pi_{\mathbf{x}}$ is given by $V_{\mathbf{x}}(\mathbf{s}_0) \coloneqq \mathbb{E}_{\mathbf{a}_0 \sim \pi_{\mathbf{x}}(\mathbf{a} \mid \mathbf{s}_0)}[Q_{\mathbf{x}}(\mathbf{s}_0, \mathbf{a}_0)]$.

The agent's problem is to learn a policy that maximises the value given an expectation over $\mathbf{s}_0$, defined either by an initial state distribution in the episodic setting (e.g. Atari, Section 5.2) or the stationary state-visitation distribution under the policy in the non-episodic setting (Section 5.1). Central to RL is the notion of *policy-improvement*, which takes a current policy $\pi_{\mathbf{x}}$ and constructs a new policy $\pi_{\mathbf{x}'}$ such that $\mathbb{E}[V_{\mathbf{x}'}] \geq \mathbb{E}[V_{\mathbf{x}}]$. A common policy-improvement step is $\arg\max_{\mathbf{x}'} \mathbb{E}_{a \sim \pi_{\mathbf{x}'}(a|s)}[Q_{\mathbf{x}}(s, a)]$.

Most works in meta-RL rely on actor-critic algorithms [56]. These treat the above policy-improvement step as an optimisation problem and estimate a *policy-gradient* [64, 56] to optimise $\mathbf{x}$. To estimate $V_{\mathbf{x}}$, these introduce a *critic* $v_{\mathbf{z}}$ that is jointly trained with the policy. The policy is optimised under

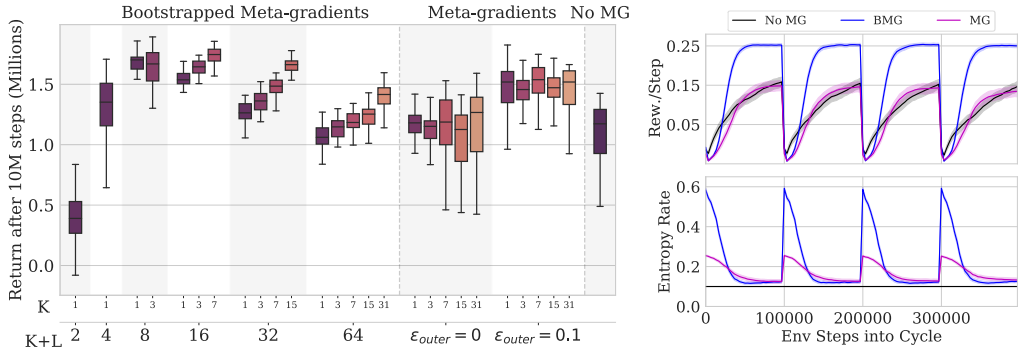

Figure 2: Non-stationary grid-world (Section 5.1). Left: Comparison of total returns under an actor-critic agent over 50 seeds. Right: Learned entropy-regularization schedules. The figure depicts the average regularization weight (ϵ) over 4 task-cycles at 6M steps in the environment.

the current estimate of its value function, while the critic is tracking the value function by minimizing a Temporal-Difference (TD) error. Given a *rollout* $\tau = (\mathbf{s}_0, \mathbf{a}_0, r_1, \mathbf{s}_1, \ldots, r_T, \mathbf{s}_T)$, the objective is given by $f(\mathbf{x}, \mathbf{z}) = \epsilon_{\text{PG}}\, \ell_{\text{PG}}(\mathbf{x}) + \epsilon_{\text{EN}}\, \ell_{\text{EN}}(\mathbf{x}) + \epsilon_{\text{TD}}\, \ell_{\text{TD}}(\mathbf{z})$, $\epsilon_{\text{PG}}, \epsilon_{\text{EN}}, \epsilon_{\text{TD}} \in \mathbb{R}_+$, where

$$\begin{aligned} \ell_{\text{EN}}(\mathbf{x}) &= -\sum_{t\in\tau}\sum_{\mathbf{a}\in\mathcal{A}} \pi_{\mathbf{x}}(\mathbf{a} \mid \mathbf{s}_t) \log \pi_{\mathbf{x}}(\mathbf{a} \mid \mathbf{s}_t), \qquad \ell_{\text{TD}}(\mathbf{z}) = \frac{1}{2}\sum_{t\in\tau}\left(G_t^{(n)} - v_{\mathbf{z}}(\mathbf{s}_t)\right)^2, \\ \ell_{\text{PG}}(\mathbf{x}) &= -\sum_{t\in\tau} \rho_t \log \pi_{\mathbf{x}}(\mathbf{a}_t \mid \mathbf{s}_t)\left(G_t^{(n)} - v_{\mathbf{z}}(\mathbf{s}_t)\right), \end{aligned} \tag{5}$$

where ρ_t denotes an importance weight and $G_t^{(n)}$ denotes an n-step bootstrap target. Its form depends on the algorithm; in Section 5.1, we generate rollouts from $\pi_{\mathbf{x}}$ (on-policy), in which case $\rho_t = 1$ and $G_t^{(n)} = \sum_{i=0}^{(n-1)} \gamma^i r_{t+i+1} + \gamma^n v_{\bar{\mathbf{z}}}(\mathbf{s}_{t+n})\, \forall t$, where $\bar{\mathbf{z}}$ denotes fixed (non-differentiable) parameters. In the off-policy setting (Section 5.2), ρ corrects for sampling bias and $G_t^{(n)}$ is similarly adjusted.

5.1 A non-stationary and non-episodic grid world

We begin with a tabular grid-world with two items to collect. Once an item is collected, it is randomly re-spawned. One item yields a reward of $+1$ and the other a reward of -1. The reward is flipped every 100,000 steps. To succeed, a memory-less agent must efficiently re-explore the environment. We study an on-policy actor-critic agent with $\epsilon_{\text{PG}} = \epsilon_{\text{TD}} = 1$. As baseline, we tune a fixed entropy-rate weight $\epsilon = \epsilon_{\text{EN}}$. We compare against agents that meta-learn ϵ online. For MG, we use the actor-critic loss as meta-objective (ϵ fixed), as per Eq. 1. The setup is described in full in Appendix B.1

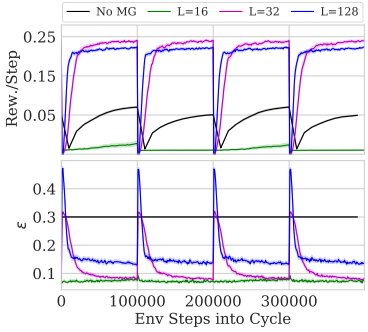

Figure 3: BMG ε-greedy exploration under a $Q(\lambda)$-agent.

BMG Our primary focus is on the effect of bootstrapping. Because this setup is fully online, we can generate targets using the most recent $L-1$ parameter updates (which does not require additional computation) and a final agent parameter update using $\epsilon = 0$. Hence, the computational complexity of BMG is constant in L under this implementation; see Appendix B.2 for implementation details. We define the matching function as the KL-divergences between $\mathbf{x}^{(K)}$ and the target, $\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}(\mathbf{w})) = \text{KL}\left(\pi_{\tilde{\mathbf{x}}} \parallel \pi_{\mathbf{x}^{(K)}}\right)$.

Figure 2 presents our main findings. Both MG and BMG learn adaptive entropy-rate schedules that outperform the baseline. However, MG fail if $\epsilon = 0$ in the meta-objective, as it fails to rapidly increase ϵ for efficient task adaptation (Appendix B.2, Figure 9). MG show no clear benefit of longer meta-learning horizons, indicating that myopia stems from the objective itself. In contrast, BMG exhibits greater adaptive capacity and is able to utilise greater meta-learning horizons. Too

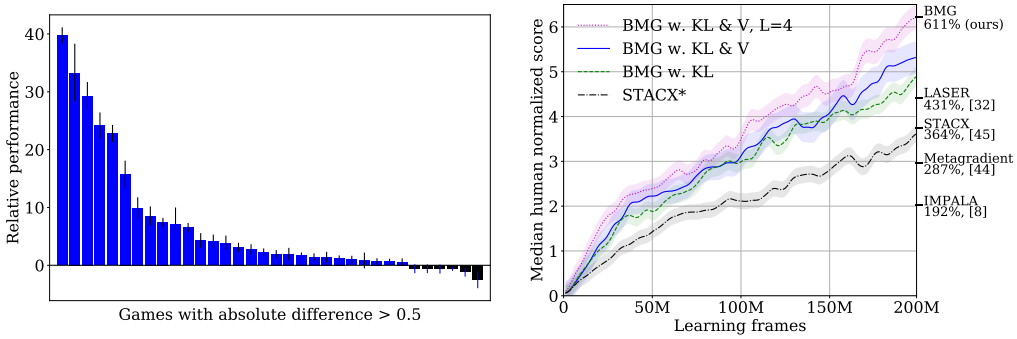

Figure 4: Human-normalized score across the 57 games in Atari ALE [8]. Left: per-game difference in score between BMG and our implementation of STACX* at 200M frames. Right: Median scores over learning compared to published baselines. Shading depict standard deviation across 3 seeds.

short horizons induce myopia, whereas too long prevent efficient adaptation. For a given horizon, increasing K is uniformly beneficial. Finally, we find that BMG outperforms MG for a given horizon without backpropagating through all updates. For instance, for $K = 8$, BMG outperforms MG with $K = 1$ and $L = 7$. We perform ablation studies in Appendix B.2; increasing the horizon in the target bootstrap effectively counters myopia, but using the meta-learned update rule for all L steps leads to positive feedback loops that can derail meta-optimization.

Next, we consider a new form of meta-learning: learning ε-greedy exploration in a $Q(\lambda)$-agent (precise formulation in Appendix B.3). While the ε parameter has a similar effect to entropy-regularization, ε is a parameter applied in the behavior-policy while acting. As it does not feature in the loss function, it is not readily optimized by existing meta-gradient approaches. In contrast, BMG can be implemented by matching the policy derived from a target action-value function, precisely as in the actor-critic case. An implication is that BMG can meta-learn *without* backpropagating through the update rule. Significantly, this opens up to meta-learning (parts of) the *behaviour policy*, which is hard to achieve in the MG setup as the behaviour policy is not used in the update rule. Figure 3 shows that meta-learning ε-greedy exploration in this environment significantly outperforms the best fixed ε found by hyper-parameter tuning. As in the actor-critic case, we find that BMG responds positively to longer meta-learning horizons (larger L); see Appendix B.3, Figure 12 for detailed results.

5.2 Atari

High-performing RL agents tend to rely on distributed learning systems to improve data efficiency [29, 15]. This presents serious challenges for meta-learning as the policy gradient becomes noisy and volatile due to off-policy estimation [66, 68]. Theorem 1 suggests that BMG can be particularly effective in this setting under the appropriate distance function. To test these predictions, we adapt the Self-Tuning Actor-Critic [STACX; 68] to meta-learn under BMG on the 57 environments in the Atari Arcade Learning Environment [ALE; 8].

Protocol We follow the original IMPALA setup [15], but we do not downsample or gray-scale inputs. Following the literature, we train for 200 million frames and evaluate agent performance by median Human Normalized Score (HNS) across 3 seeds [15, 66, 68]. Statistical tests use a sample size of 57×3 due to independence of environments. For experimental details, see Appendix C.

STACX The IMPALA actor-critic agent runs multiple actors asynchronously to generate experience for a centralized learner. The learner uses truncated importance sampling to correct for off-policy data in the actor-critic update, which adjusts ρ and $\hat{V}$ in Eq. 5. The STACX agent [68] is a state-of-the-art meta-RL agent. It builds on IMPALA in two ways: (1) it introduces auxiliary tasks in the form of additional objectives that differ only in their hyper-parameters; (2) it meta-learns the hyper-parameters of each loss function (main and auxiliary). Meta-parameters are given by $\mathbf{w} = (\gamma^i, \epsilon^i_{\text{PG}}, \epsilon^i_{\text{EN}}, \epsilon^i_{\text{TD}}, \lambda^i, \alpha^i)_{i=1}^{1+n}$, where λ and α are hyper-parameters of the importance weighting mechanism and $n = 2$ denotes the number of auxiliary tasks. STACX uses the IMPALA objective as the meta-objective with $K = 1$. The same batch of rollouts is used in the agent parameter update and

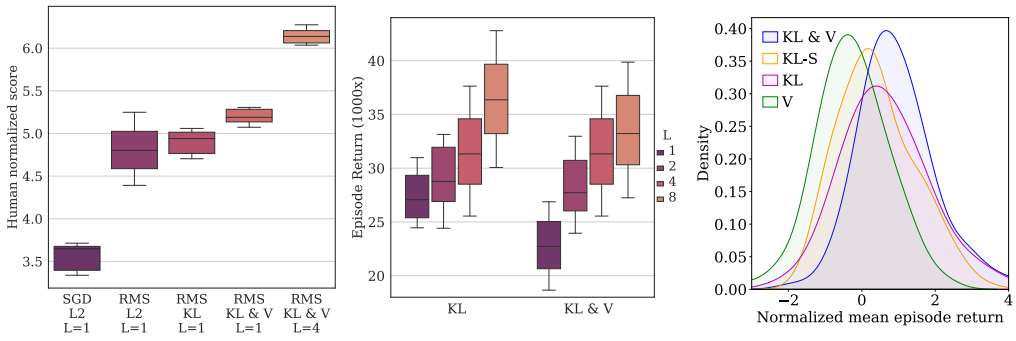

Figure 5: Ablations on Atari. Left: human normalized score decomposition of TB w.r.t. optimizer (SGD, RMS), matching function (L2, KL, KL & V), and bootstrap steps (L). BMG with (SGD, L2, $L = 1$) is equivalent to STACX. Center: episode return on Ms Pacman for different L. Right: distribution of episode returns over all 57 games, normalized per-game by mean and standard deviation. All results are reported between 190-200M frames over 3 independent seeds.

in the MG update. Our implementation of STACX use an identical setup, except for a slightly larger network and 60% experience replay to compensate for a richer input space; it matches the median HNS reported in [68]. See Appendix C for a complete description.

BMG We conduct ceteris-paribus comparisons that only alter the meta-objective: agent parameter updates are *identical* to those in STACX. When $L = 1$, the target takes a gradient step on the original IMPALA loss. As in STACX, we use the same batch of rollouts in the meta-update as in the agent parameter update. Hence the *only* difference is the form of the meta-objective; BMG does not use more data nor more gradient information. For $L > 1$, the first $L - 1$ steps bootstrap from the meta-learned update rule itself. To avoid overfitting, each of the $L - 1$ steps use separate replay data; this extra data is not used anywhere else. To understand matching functions, we test policy matching and value matching. Policy matching is defined by $\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}(\mathbf{w})) = \mathrm{KL}\left(\pi_{\tilde{\mathbf{x}}} \parallel \pi_{\mathbf{x}^{(1)}}\right)$; we also test a symmetric KL-divergence (KL-S). Value matching is defined by $\mu(\tilde{\mathbf{z}}, \mathbf{z}^{(1)}(\mathbf{w})) \coloneqq \mathbb{E}\left[\left(v_{\tilde{z}} - v_{z^{(1)}}\right)^2\right]$.

Figure 4 presents our main comparison. BMG with $L = 1$ and policy-matching (KL) obtains a median HNS of ~500%, compared to ~350% for STACX. Recall that for $L = 1$, BMG uses the same data to compute agent parameter update, target update, and matching loss; hence this is an apples-to-apples comparison. Using both policy matching and value matching (with 0.25 weight on the latter) further improves the score to ~520% and outperforms STACX across almost all 57 games, with a few minor exceptions (left panel, Figure 4). These results are obtained *without* tuning hyper-parameters for BMG. Finally, extending the meta-learning horizon by setting $L = 4$ and adjusting gradient clipping from .3 to .2 obtains a score of ~610%.

In Figure 5, we turn to ablations. In the left-panel, we deconstruct BMG into STACX (i.e., MG) and compare performances. We find that roughly 45% of the performance gains comes from curvature correction (given by using RMSProp in the target bootstrap). The matching function can further control curvature to obtain performance improvements, accounting for roughly 25%. Finally, increasing L accounts for about 30% of the performance improvement. Taken together, these results indicate that about 70% of the gain stems from improving curvature and 30% stems from reducing myopia. Comparing the cosine similarity between consequtive meta-gradients, we find that BMG improves upon STACX by two orders of magnitude. For full details, see Appendix C.1.

The center panel of Figure 5 provides a deep-dive in the effect of increasing the meta-learning horizon ($L > 1$) in Ms Pacman. We find a clear pattern of increased performance in L. Recall that that $L - 1$ steps are taken under the meta-learner; only the final step uses the original IMPALA objective. These results further support that BMG can increase the effective meta-horizon in challenging meta-learning problem without increasing the number of update steps to backpropagate through. We provide further in-depth analysis of the meta-learning horizon in Appendix C.3 and find that increasing K is fundamentally different from increasing L. In particular, increasing K is more sensitive to the quality of data and ill-conditioning. An important finding is that *only* bootstrapping from the meta-learner for all L steps results in a complete breakdown (Appendix C.2, Figure 14). We ablate the role of

replay in meta-updates Appendix C.2. We find that standard MG generally degrades as we increase the amount of replay, but that BMG can successfully leverage replay in the target bootstrap.

The right panel of Figure 5 studies the effect of the matching function. Overall, we find that joint policy and value matching exhibits superior performance. Interestingly, while recent work [58, 22] suggest that reversing or using a symmetric KL-objective can yield improvements, we do not find this to be the case here. Using only value-matching results in worse performance, as it only optimises for efficient policy evaluation but lacks an explicit signal for optimising the policy improvement operator. Finally, we conduct detailed analysis of scalability. We find that while BMG is 20% slower for $K = 1, L = 1$ due to the target bootstrap, it is 200% faster when MG uses $K = 4$ and BMG uses $K = 1, L = 3$. For more detailed results, including effect of network size, see Appendix C.4.

6 Multi-Task Few-Shot Learning

Multi-task meta-learning introduces an expectation over task objectives. This is straight-forward to incorporate with BMG, as we can compute task-specific bootstrap targets and compute the meta-gradient in expectation over the task-specific matching losses. We provide a general formulation for multi-task meta-learning with MG and BMG in Appendix D; here we focus on the few-shot classification paradigm. Let $f_{\mathcal{D}} : \mathcal{X} \to \mathbb{R}$ denote the negative log-likelihood loss on some data $\mathcal{D}$. A *task* is defined as a pair of datasets $(\mathcal{D}_\tau, \mathcal{D}'_\tau)$, where $\mathcal{D}_\tau$ is a training set and $\mathcal{D}'_\tau$ is a validation set. In the M-shot-N-way setting, each task has N classes and $\mathcal{D}_\tau$ contains M observations per class.

The goal of this study is to demonstrate that BMG can serve as a plug-in replacement of the meta-objective in the multi-task setting, and to uncover interesting implications of using a KL-based matching function in the classification setting. The canonical MG method in this setting is MAML [16], which meta-learns an initialisation $\mathbf{x}_\tau^{(0)} = \mathbf{w}$ for SGD that is shared across a task distribution $p(\tau)$. Adaptation is defined by $\mathbf{x}_\tau^{(k)} = \mathbf{x}_\tau^{(k-1)} + \alpha \nabla f_{\mathcal{D}_\tau}(\mathbf{x}_\tau^{(k-1)})$, with $\alpha \in \mathbb{R}_+$ fixed. MAML uses the MG update where the meta-objective is the validation loss in expectation over the task distribution: $\mathbb{E}[f_{\mathcal{D}'_\tau}(\mathbf{x}_\tau^{(K)}(\mathbf{w}))]$. Several works have extended the MAML framework by altering the inner update φ to also control curvature in the *inner* optimization problem [31, 70, 40, 18]. As they all rely on the MG update, we focus on apples-to-apples comparisons with MAML.

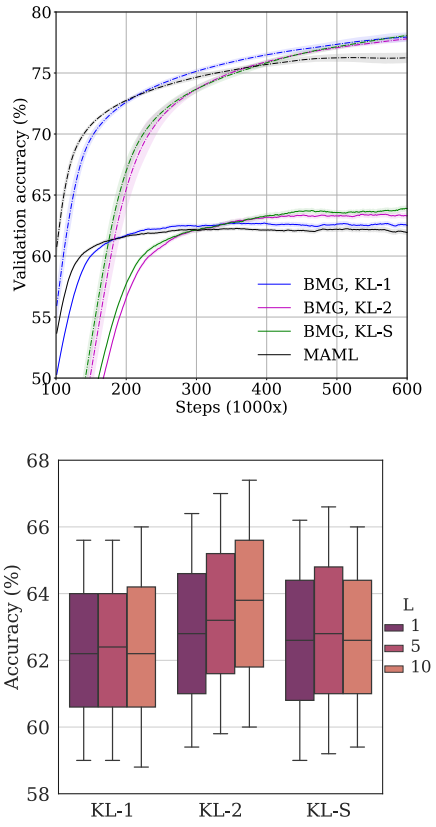

Figure 6: MiniImageNet 5-way-5-shot. Top: training (dashed) and validation (solid) curves for $K = 5, L = 10$; shading represent standard deviation over 5 seeds. Bottom: ablation over matching function and L on validation set.

BMG Similarly to MG, the BMG multi-task loss is the expectation over task-specific target matching losses. Specifically, we generate task-specific targets by taking a further L SGD steps from task-specific parameters $\mathbf{x}_\tau^{(K)}$ using *validation data*. Given the target $\tilde{\mathbf{x}}_\tau$, we compute the task-specific matching loss $\mu(\tilde{\mathbf{x}}_\tau, \mathbf{x}_\tau^{(K)})$ and, as in MG, compute the meta-objective in expectation over the task distribution: $\mathbb{E}[\mu(\tilde{\mathbf{x}}_\tau, \mathbf{x}_\tau^{(K)})]$. We focus on KL-divergences as the matching function, which has an interesting link to MG; $\tilde{\mathbf{x}}_\tau$ can be seen as an "expert" on task τ, produced by training on held-out data. The KL-divergence forms a distillation loss [23] between this expert and the task learner $\mathbf{x}_\tau^{(K)}$. MAML can also be cast in this light by noting that the log-likelihood loss is the KL divergence between a "cold" expert that places all mass on the true label. We hypothesise that, because BMG is distilling towards a distribution with higher temperature, its update can transfer more information than MAML [23].

Table 1: M-shot-5-way test accuracy on MiniImagenet. $\pm$ denotes standard deviation over 5 seeds.

	BMG, KL-1	BMG, KL-2	BMG, KL-s	MAML	FO-MAML
1-shot	$49.1 \pm .5$	**50.2** $\pm 1.$	$49.3 \pm .4$	$49.2 \pm .7$	$47.9 \pm .1$
5-shot	$64.2 \pm .4$	**65.0** $\pm 1.$	$64.2 \pm .3$	$63.3 \pm .3$	$62.6 \pm .1$

We test these predictions on the MiniImagenet benchmark [61], following the original protocol, but we train for 600 000 steps with a $10\times$ lower learning rate as we found this to give better performance. We compare three matching functions: KL-1 is mode-covering, $\mathrm{KL}\left(\pi_{\tilde{\mathbf{x}}} \parallel \pi_{\mathbf{x}^{(K)}}\right)$, while KL-2 is mode-seeking, $\mathrm{KL}\left(\pi_{\mathbf{x}} \parallel \pi_{\tilde{\mathbf{x}}}\right)$; KL-S averages the two. We follow the original protocol for MAML and set $K = 5$, with $L \in \{1, 5, 10\}$ for BMG. We also report scores for the first-order approximation to MAML [FO-MAML; 16] that avoids backpropagating through all steps, to compare against a version of MG that scales similarly to BMG in the meta-learning horizon.

Table 1 report accuracy on the meta-test set. We use $K = 5$ and $L = 10$. FO-MAML take 10 gradient steps. At meta-test time, all methods adapt to a task by taking 10 gradient steps, as per the original protocol [16]. We find that BMG can indeed improve upon the performance of MAML, particularly in the 5-shot setting where target updates have more data. Remarkably, even in the 1-shot setting, where the target is taking gradient steps *on a single held-out sample*, BMG slightly outperforms MAML. While FO-MAML allows longer horizons, its first-order approximation degrades performance.

Figure 6 ablates BMG design choices. Overall, we find KL-2 to perform best due to its entropy regularizing effect, as evidenced by taking longer to overfit the training set. KL-S is a close second, while KL-1 suffers from overfitting. While KL-1 and KL-S do not show a benefit from increasing the meta-learning horizon, again indicating overfitting, KL-2 does improve its performance on held-out task as we increase L. In summary, BMG can increase the effective meta-learning horizon without increasing the number of updates to backpropagate through if the meta-learner is not overfitting.

We conduct an analysis of the impact BMG has on curvature and meta-gradient variance in Appendix D.3. Gains in terms of ill-conditioning are smaller in this setting, as the MAML loss is already a form of KL-divergence. However, curvature improves as the bootstrapping horizon is increased, suggesting that BMG is indeed able to transfer more information, in terms of generalization performance. The BMG meta-gradient exhibits smaller variance than MAML, with an order of magnitude larger meta-gradient norm to variance ratio. In terms of scalability, BMG with $L = 1$ is roughly on par with MAML, while for $L = 10$, it is 50% slower. In contrast, for MAML to attain the same effective meta-gradient horizon, it would be 56% slower. See Appendix D.4 for full details.

7 Conclusion

In this paper, we have put forth the notion that efficient meta-learning does not require the meta-objective to be expressed directly in terms of the learner's objective. Instead, we present an alternative approach that relies on target matching. The central tenet is to compare the output of the meta-learner with a desired target that the meta-learner should have produced. While such targets can be produced in a variety of ways, an appealing approach is produce a target by bootstrapping from some update rule–either the one being meta-learned or another update rule. That is, to keep updating the learner's parameter under this bootstrapped update rule and use these "future" parameters as the target. While using the meta-learned update rule as the bootstrap allows for an open-ended meta-learning process, we find that some grounding in the learner's objective is necessary.

As an instance of this approach, we study bootstrapped meta-gradients. The algorithm bootstraps a target for the meta-learner to match by taking L gradient update steps under some (mix of) update rule(s) and match the meta-learner's output after K steps of updates against this target. This form of meta-learning can guarantee performance improvements under appropriate choices of targets and matching function, and can guarantee larger improvements than under standard meta-gradients. Empirically, we observe substantial improvements on Atari, where we achieve a new state-of-the-art. We also find that bootstrapped meta-gradients works well in multi-task meta-learning, where it improves upon MAML. Finally, we explore new possibilities afforded by the target-matching nature of the algorithm, and demonstrate that it can learn to explore in an ϵ-greedy Q-learning agent.

Acknowledgments and Disclosure of Funding

The authors would like to thank Guillaume Desjardins, Junhyuk Oh, Luisa Zintgraf, Razvan Pascanu, and Nando de Freitas for insightful feedback on earlier versions of this paper. This work was funded by DeepMind.

References

[1] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a Posteriori Policy Optimisation. In *International Conference on Learning Representations*, 2018.

[2] F. Alet, M. F. Schneider, T. Lozano-Perez, and L. P. Kaelbling. Meta-Learning Curiosity Algorithms. In *International Conference on Learning Representations*, 2020.

[3] S.-I. Amari. Natural Gradient Works Efficiently in Learning. *Neural computation*, 10(2):251–276, 1998.

[4] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to Learn by Gradient Descent by Gradient Descent. In *Advances in Neural Information Processing Systems*, 2016.

[5] A. Antoniou, H. Edwards, and A. J. Storkey. How to Train Your MAML. In *International Conference on Learning Representations*, 2019.

[6] M.-F. Balcan, M. Khodak, and A. Talwalkar. Provable Guarantees for Gradient-Based Meta-Learning. In *International Conference on Machine Learning*, 2019.

[7] A. Beck and M. Teboulle. Mirror Descent and Nonlinear Projected Subgradient Methods for Convex Optimization. *Operations Research Letters*, 31:167–175, 2003.

[8] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

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

[10] Y. Bengio, S. Bengio, and J. Cloutier. *Learning a Synaptic Learning Rule*. Université de Montréal, Département d'informatique et de recherche opérationnelle, 1991.

[11] Y. Cao, T. Chen, Z. Wang, and Y. Shen. Learning to Optimize in Swarms. *Advances in Neural Information Processing Systems*, 2019.

[12] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, and N. de Freitas. Learning to learn for Global Optimization of Black Box Functions. In *Advances in Neural Information Processing Systems*, 2016.

[13] G. Denevi, D. Stamos, C. Ciliberto, and M. Pontil. Online-Within-Online Meta-Learning. In *Advances in Neural Information Processing Systems*, 2019.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A Large-Scale Hierarchical Image Database. In *Computer Vision and Pattern Recognition*, 2009.

[15] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *International Conference on Machine Learning*, 2018.

[16] C. Finn, P. Abbeel, and S. Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning*, 2017.

[17] S. Flennerhag, P. G. Moreno, N. D. Lawrence, and A. Damianou. Transferring Knowledge across Learning Processes. In *International Conference on Learning Representations*, 2019.

[18] S. Flennerhag, A. A. Rusu, R. Pascanu, F. Visin, H. Yin, and R. Hadsell. Meta-Learning with Warped Gradient Descent. In *International Conference on Learning Representations*, 2020.

[19] E. Grant, C. Finn, S. Levine, T. Darrell, and T. L. Griffiths. Recasting Gradient-Based Meta-Learning as Hierarchical Bayes. In *International Conference on Learning Representations*, 2018.

[20] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, and M. Valko. Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning. In *Advances in Neural Information Processing Systems*, 2020.

[21] Z. D. Guo, B. A. Pires, B. Piot, J.-B. Grill, F. Altché, R. Munos, and M. G. Azar. Bootstrap Latent-Predictive Representations for Multitask Reinforcement Learning. In *International Conference on Machine Learning*, 2020.

[22] M. Hessel, I. Danihelka, F. Viola, A. Guez, S. Schmitt, L. Sifre, T. Weber, D. Silver, and H. van Hasselt. Muesli: Combining Improvements in Policy Optimization. *arXiv preprint arXiv:2104.06159*, 2021.

[23] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] G. E. Hinton and D. C. Plaut. Using Fast Weights to Deblur Old Memories. In *Cognitive Science Society*, 1987.

[25] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning To Learn Using Gradient Descent. In *International Conference on Artificial Neural Networks*, 2001.

[26] G. Jerfel, E. Grant, T. Griffiths, and K. A. Heller. Reconciling Meta-Learning and Continual Learning with Online Mixtures of Tasks. In *Advances in Neural Information Processing Systems*, 2019.

[27] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, and A. Borchers. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *International Symposium on Computer Architecture*, 2017.

[28] S. M. Kakade. A Natural Policy Gradient. In *Advances in Neural Information Processing Systems*, 2001.

[29] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent Experience Replay in Distributed Reinforcement Learning. In *International Conference on Learning Representations*, 2018.

[30] M. Khodak, M.-F. F. Balcan, and A. S. Talwalkar. Adaptive Gradient-Based Meta-Learning Methods. *Advances in Neural Information Processing Systems*, 2019.

[31] Y. Lee and S. Choi. Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace. In *International Conference on Machine Learning*, 2018.

[32] K. Lv, S. Jiang, and J. Li. Learning Gradient Descent: Better Generalization and Longer Horizons. In *International Conference on Machine Learning*, 2017.

[33] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[34] D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-Based Hyperparameter Optimization Through Reversible Learning. In *International conference on machine learning*, pages 2113–2122. PMLR, 2015.

[35] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein. Understanding and Correcting Pathologies in the Training of Learned Optimizers. In *International Conference on Machine Learning*, 2019.

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

[37] Y. Nesterov. A Method for Solving the Convex Programming Problem with Convergence Rate $O(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

[38] A. Nichol, J. Achiam, and J. Schulman. On First-Order Meta-Learning Algorithms. *arXiv preprint ArXiv:1803.02999*, 2018.

[39] J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. P. van Hasselt, S. Singh, and D. Silver. Discovering Reinforcement Learning Algorithms. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[40] E. Park and J. B. Oliva. Meta-Curvature. In *Advances in Neural Information Processing Systems*, 2019.

[41] R. Pascanu and Y. Bengio. Revisiting Natural Gradient for Deep Networks. In *International Conference on Learning Representations*, 2014.

[42] J. Peng and R. J. Williams. Incremental Multi-Step Q-Learning. In *International Conference on Machine Learning*, 1994.

[43] S. Ravi and H. Larochelle. Optimization as a Model for Few-Shot Learning. In *International Conference on Learning Representations*, 2017.

[44] E. Real, C. Liang, D. R. So, and Q. V. Le. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. In *International Conference on Machine Learning*, 2020.

[45] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy Distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[46] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-Learning with Latent Embedding Optimization. In *International Conference on Learning Representations*, 2019.

[47] J. Schmidhuber. *Evolutionary Principles in Self-Referential Learning*. PhD thesis, Technische Universität München, 1987.

[48] J. Schmidhuber. A 'self-referential' weight matrix. In *International Conference on Artificial Neural Networks*, pages 446–450. Springer, 1993.

[49] S. Schmitt, M. Hessel, and K. Simonyan. Off-Policy Actor-Critic with Shared Experience Replay. In *International Conference on Machine Learning*, 2020.

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

[51] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust Region Policy Optimization. In *International Conference on Machine Learning*, 2015.

[52] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[53] E. S. Spelke and K. D. Kinzler. Core Knowledge. *Developmental science*, 10(1):89–96, 2007.

[54] 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. In *Advances in Neural Information Processing Systems*, 2018.

[55] R. S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine learning*, 3(1):9–44, 1988.

[56] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, volume 99, 1999.

[57] Y. W. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust Multitask Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2017.

[58] M. Tomar, L. Shani, Y. Efroni, and M. Ghavamzadeh. Mirror Descent Policy Optimization. *arXiv preprint arXiv:2005.09814*, 2020.

[59] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, and P.-A. Manzagol. Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples. *International Conference on Learning Representations*, 2020.

[60] T. van Erven and W. M. Koolen. MetaGrad: Multiple Learning Rates in Online Learning. In *Advances in Neural Information Processing Systems*, 2016.

[61] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching Networks for One Shot Learning. In *Advances in Neural Information Processing Systems*, 2016.

[62] 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. In *Annual Meeting of the Cognitive Science Society*, 2016.

[63] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein. Learned Optimizers that Scale and Generalize. In *International Conference on Machine Learning*, 2017.

[64] R. J. Williams and J. Peng. Function Optimization using Connectionist Reinforcement Learning Algorithms. *Connection Science*, 3(3):241–268, 1991.

[65] Y. Wu, M. Ren, R. Liao, and R. B. Grosse. Understanding Short-Horizon Bias in Stochastic Meta-Optimization. In *International Conference on Learning Representations*, 2018.

[66] Z. Xu, H. P. van Hasselt, and D. Silver. Meta-Gradient Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2018.

[67] M. Yin, G. Tucker, M. Zhou, S. Levine, and C. Finn. Meta-Learning without Memorization. In *International Conference on Learning Representations*, 2020.

[68] T. Zahavy, Z. Xu, V. Veeriah, M. Hessel, J. Oh, H. P. van Hasselt, D. Silver, and S. Singh. A Self-Tuning Actor-Critic Algorithm. *Advances in Neural Information Processing Systems*, 33, 2020.

[69] Z. Zheng, J. Oh, and S. Singh. On Learning Intrinsic Rewards for Policy Gradient Methods. *Advances in Neural Information Processing Systems*, 2018.

[70] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast Context Adaptation via Meta-Learning. In *International Conference on Machine Learning*, 2019.

Bootstrapped Meta-Learning: Appendix

Contents

A Proofs

This section provides complete proofs for the results in Section 4. Throughout, we assume that $(\mathbf{x}^{(0)}, \mathbf{h}^{(0)}, \mathbf{w})$ is given and write $\mathbf{x} \coloneqq \mathbf{x}^{(0)},\ \mathbf{h} \coloneqq \mathbf{h}^{(0)}$. We assume that $\mathbf{h}$ evolves according to some process that maps a history $H^{(k)} \coloneqq (\mathbf{x}^{(0)}, \mathbf{h}^{(0)}, \dots, \mathbf{x}^{(k-1)}, \mathbf{h}^{(k-1)}, \mathbf{x}^{(k)})$ into a new learner state $\mathbf{h}^{(k)}$, including any sampling of data (c.f. Section 3). Recall that we restrict attention to the noiseless setting, and hence updates are considered in expectation. We define the *map* $\mathbf{x}^{(K)}(\mathbf{w})$ by

$$\begin{aligned}
\mathbf{x}^{(1)} &= \mathbf{x}^{(0)} + \varphi\big(\mathbf{x}^{(0)}, \mathbf{h}^{(0)}, \mathbf{w}\big) \\
\mathbf{x}^{(2)} &= \mathbf{x}^{(1)} + \varphi\big(\mathbf{x}^{(1)}, \mathbf{h}^{(1)}, \mathbf{w}\big) \\
&\vdots \\
\mathbf{x}^{(K)} &= \mathbf{x}^{(K-1)} + \varphi\big(\mathbf{x}^{(K-1)}, \mathbf{h}^{(K-1)}, \mathbf{w}\big).
\end{aligned}$$

The derivative $\frac{\partial}{\partial \mathbf{w}} \mathbf{x}^{(K)}(\mathbf{w})$ differentiates through each step of this process [25]. As previously stated, we assume f is Lipschitz and that $\mathbf{x}^{(K)}$ is Lipschitz w.r.t. $\mathbf{w}$; we assume that μ satisfies Eq. 4. We are now in a position to prove results from the main text. We re-state them for convenience.

Lemma 1 (MG Descent)**.** *Let $\mathbf{w}'$ be given by Eq. 1. For β sufficiently small, $f\big(\mathbf{x}^{(K)}(\mathbf{w}')\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) = -\beta \|\nabla_x f(\mathbf{x}^{(K)})\|^2_{G^T} + O(\beta^2) < 0$.*

Proof. Define $\mathbf{g} \coloneqq \nabla_x f(\mathbf{x}^{(K)}(\mathbf{w}))$. The meta-gradient at $(\mathbf{x}, \mathbf{h}, \mathbf{w})$ is given by $\nabla_w f(\mathbf{x}^{(K)}(\mathbf{w})) = D\,\mathbf{g}$. Under Eq. 1, we find $\mathbf{w}' = \mathbf{w} - \beta D\,\mathbf{g}$. By first-order Taylor Series Expansion of f around $(\mathbf{x}, \mathbf{h}, \mathbf{w}')$ with respect to $\mathbf{w}$:

$$\begin{aligned}
f\big(\mathbf{x}^{(K)}(\mathbf{w}')\big) &= f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) + \langle D\,\mathbf{g}, \mathbf{w}' - \mathbf{w}\rangle + O(\|\,\mathbf{w}' - \mathbf{w}\,\|_2^2) \\
&= f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) - \beta\langle D\,\mathbf{g}, D\,\mathbf{g}\rangle + O(\beta^2) \\
&= f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) - \beta\|\,\mathbf{g}\,\|^2_{G^T} + O(\beta^2).
\end{aligned}$$

Noting that $\|\,\mathbf{g}\,\|^2_{G^T} \geq 0$ by virtue of positive semi-definiteness of G completes the proof. ■

Theorem 1 (BMG Descent)**.** *Let $\tilde{\mathbf{w}}$ be given by Eq. 2 with target $\tilde{\mathbf{x}} = \xi^{\alpha}_{G}(\mathbf{x}^{(K)})$. For β sufficiently small, $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) < 0$. Further, for α, β sufficiently small, $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) = -\frac{\beta}{\alpha}\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) + O(\beta(\alpha+\beta)) < 0$.*

Proof. The bootstrapped meta-gradient at $(\mathbf{x}, \mathbf{h}, \mathbf{w})$ is given by

$$\nabla_w \mu\Big(\tilde{\mathbf{x}},\, \mathbf{x}^{(K)}(\mathbf{w})\Big) = D\,\mathbf{u}, \quad \text{where} \quad \mathbf{u} \coloneqq \nabla_z \mu\big(\tilde{\mathbf{x}},\, \mathbf{z}\big)\Big|_{\mathbf{z}=\mathbf{x}^{(K)}}.$$

Under Eq. 2, we find $\tilde{\mathbf{w}} = \mathbf{w} - \beta D\,\mathbf{u}$. Define $\mathbf{g} \coloneqq \nabla_x f(\mathbf{x}^{(K)})$. By first-order Taylor Series Expansion of f around $(\mathbf{x}, \mathbf{h}, \tilde{\mathbf{w}})$ with respect to $\mathbf{w}$:

$$
\begin{aligned}
f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) &= f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) + \langle D\,\mathbf{g}, \tilde{\mathbf{w}} - \mathbf{w}\rangle + O(\|\tilde{\mathbf{w}} - \mathbf{w}\|_2^2) \\
&= f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) - \beta\langle D\,\mathbf{g}, D\,\mathbf{u}\rangle + O(\beta^2) \\
&= f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) - \beta\langle \mathbf{u}, G^T\,\mathbf{g}\rangle + O(\beta^2). \qquad (6)
\end{aligned}
$$

To bound the inner product, by convexity in $\mu(\tilde{\mathbf{x}}, \cdot)$:

$$
\begin{aligned}
\mu(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}) &\geq \mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) + \left\langle \mathbf{u},\, \tilde{\mathbf{x}} - \mathbf{x}^{(K)} \right\rangle \\
&\geq \mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) + \left\langle \mathbf{u},\, \mathbf{x}^{(K)} - \alpha G^T\,\mathbf{g} - \mathbf{x}^{(K)} \right\rangle \\
&\geq \mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) - \alpha \left\langle \mathbf{u},\, G^T\,\mathbf{g} \right\rangle.
\end{aligned}
$$

Using that $\mu(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}) = 0$ by virtue Eq. 4, we find $-\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) \geq -\alpha\langle \mathbf{g}, G^T\,\mathbf{u}\rangle$. Further, from Eq. 4, we have that $\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) > 0$. Therefore, $-\langle \mathbf{g}, G^T\,\mathbf{u}\rangle < 0$. Thus, for β sufficiently small the residual vanishes and $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) < 0$, as was to be proved. For the final part of the proof, from a first-order Taylor series expansion of $\mu(\tilde{\mathbf{x}}, \cdot)$, we have $-\frac{\beta}{\alpha}\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) = -\beta\langle \mathbf{u}, G^T\,\mathbf{g}\rangle + O(\alpha\beta)$. Then, in Eq. 6, substitute $-\beta\left\langle \mathbf{u},\, G^T\,\mathbf{g}\right\rangle$ for $-\frac{\beta}{\alpha}\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) + O(\alpha\beta)$. ■

We now prove that, controlling for scale, BMG can yield larger performance gains than MG. Recall that $\xi_G^{\alpha}(\mathbf{x}^{(K)}) = \mathbf{x}^{(K)} - \alpha G^T \nabla f\,\mathbf{x}^{(K)}$. Consider ξ_G^r, with $r \coloneqq \|\nabla f(\mathbf{x}^{(K)})\|_2 / \|G^T \nabla f(\mathbf{x}^{(K)})\|_2$.

Corollary 1. *Let $\mu = \|\cdot\|_2^2$ and $\tilde{\mathbf{x}} = \xi_G^r(\mathbf{x}^{(K)})$. Let $\mathbf{w}'$ be given by Eq. 1 and $\tilde{\mathbf{w}}$ be given by Eq. 2. For β sufficiently small, $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) \leq f\big(\mathbf{x}^{(K)}(\mathbf{w}')\big)$, with strict inequality if $GG^T \neq G^T$.*

Proof. Let $\mathbf{g} \coloneqq \nabla_x f\big(\mathbf{x}^{(K)}\big)$. By Lemma 1, $f\big(\mathbf{x}^{(K)}(\mathbf{w}')\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) = -\beta\langle G^T\,\mathbf{g}, \mathbf{g}\rangle + O(\beta^2)$. From Theorem 1, with $\mu = \|\cdot\|_2^2$, $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) = -r\langle G^T\,\mathbf{g}, G^T\,\mathbf{g}\rangle + O(\beta(\alpha+\beta))$. For β sufficiently small, the inner products dominate and we have

$$
f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w}')\big) \approx -\beta\left(r\langle G^T\,\mathbf{g}, G^T\,\mathbf{g}\rangle - \langle G^T\,\mathbf{g}, \mathbf{g}\rangle\right).
$$

To determine the sign of the expression in parenthesis, consider the problem

$$
\max_{\mathbf{v}\in\mathbb{R}^{n_x}} \langle G^T\,\mathbf{g}, \mathbf{v}\rangle \qquad \text{s.t.} \quad \|\,\mathbf{v}\,\|_2 \leq 1.
$$

Form the Lagrangian $\mathcal{L}(\mathbf{v}, \lambda) \coloneqq \langle G^T\,\mathbf{g}, \mathbf{v}\rangle - \lambda(\|\,\mathbf{v}\,\|_2 - 1)$. Solve for first-order conditions:

$$
G^T\,\mathbf{g} - \lambda\frac{\mathbf{v}^*}{\|\,\mathbf{v}^*\,\|_2} = 0 \implies \mathbf{v}^* = \frac{\|\,\mathbf{v}^*\,\|_2}{\lambda} G^T\,\mathbf{g}.
$$

If $\lambda = 0$, then we must have $\|\,\mathbf{v}^*\,\|_2 0$, which clearly is not an optimal solution. Complementary slackness then implies $\|\,\mathbf{v}^*\,\|_2 = 1$, which gives $\lambda = \|\,\mathbf{v}^*\,\|_2 \|G^T\,\mathbf{g}\,\|_2$ and hence $\mathbf{v}^* = G^T\,\mathbf{g}\,/\|G^T\,\mathbf{g}\,\|_2$. By virtue of being the maximiser, $\mathbf{v}^*$ attains a higher function value than any other $\mathbf{v}$ with $\|\,\mathbf{v}\,\|_2 \leq 1$, in particular $\mathbf{v} = \mathbf{g}\,/\|\,\mathbf{g}\,\|_2$. Evaluating the objective at these two points gives

$$
\frac{\langle G^T\,\mathbf{g}, G^T\,\mathbf{g}\rangle}{\|G^T\,\mathbf{g}\,\|_2} \geq \frac{\langle G^T\,\mathbf{g}, \mathbf{g}\rangle}{\|\,\mathbf{g}\,\|_2} \implies r\langle G^T\,\mathbf{g}, G^T\,\mathbf{g}\rangle \geq \langle G^T\,\mathbf{g}, \mathbf{g}\rangle,
$$

where we use that $r = \|\,\mathbf{g}\,\|_2 / \|G^T\,\mathbf{g}\,\|_2$ by definition. Thus $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) \leq f\big(\mathbf{x}^{(K)}(\mathbf{w}')\big)$, with strict inequality if $GG^T \neq G^T$ and $G^T\,\mathbf{g} \neq \mathbf{0}$. ■

Corollary 2. *For any ξ, let $\tilde{\mathbf{w}}$ be given by Eq. 2. Define $\bar{\mathbf{x}} = \xi_G^{\beta}(\mathbf{x}^{(K)})$. Let μ be convex in its second argument. For β sufficiently small, if $\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) > \mu(\tilde{\mathbf{x}}, \bar{\mathbf{x}})$, then $f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) < 0$.*

Proof. Define $\mathbf{u} \coloneqq \nabla_z \mu(\tilde{\mathbf{x}}, \mathbf{z})\big|_{\mathbf{z}=\mathbf{x}^{(K)}}$. From the proof of Theorem 1, we have that for $\alpha = \beta$,

$$f\big(\mathbf{x}^{(K)}(\tilde{\mathbf{w}})\big) - f\big(\mathbf{x}^{(K)}(\mathbf{w})\big) = -\beta\langle \mathbf{u}, G^T\,\mathbf{g}\rangle + O(\beta^2) = \langle \mathbf{u}, \bar{\mathbf{x}} - \mathbf{x}^{(K)}\rangle + O(\beta^2).$$

Since $\mathbf{u}$ is the gradient of the BMG objective under $\tilde{\mathbf{x}}$, it does not immediately follow that this guarantees an improvement. From convexity of μ, we have that $\mu(\tilde{\mathbf{x}}, \bar{\mathbf{x}}) \geq \mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) + \big\langle \mathbf{u},\, \bar{\mathbf{x}} - \mathbf{x}^{(K)}\big\rangle$. Thus, if $\mu(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}) > \mu(\tilde{\mathbf{x}}, \bar{\mathbf{x}})$, it follows that $\big\langle \mathbf{u},\, \bar{\mathbf{x}} - \mathbf{x}^{(K)}\big\rangle \leq 0$. ■

B Non-Stationary non-episodic reinforcement learning

B.1 Setup

This experiment is designed to provide a controlled setting to delineate the differences between standard meta-gradients and bootstrapped meta-gradients. The environment is a 5×5 grid world with two objects; a blue and a red square (Figure 7). Thus, we refer to this environment as the *two-colors* domain. At each step, the agent (green) can take an action to move either up, down, left, or right and observes the position of each square and itself. Each position is encoded as a pair of one-hot vectors describing each coordinate. If the agent reaches a coloured square, it obtains a reward of either $+1$ or -1 while the colour is randomly moved to an unoccupied location. Every 100 000 steps, the reward for each object flips. For all other transitions, the agent obtains a reward of -0.04.

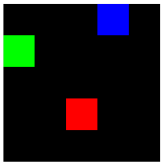

Figure 7: Two-colors Grid-world. The agent's goal is to collect either blue or red squared by navigating the green square.

The two-colors domain is designed such that the central component determining how well a memory-less agent adapts is its exploration. Our agents can only regulate exploration through policy entropy. Thus, to converge on optimal task behaviour, the agent must reduce policy entropy. Once the task switches, the agent encounters what is effectively a novel task (due to it being memory-less). To rapidly adapt the agent must first increase entropy in the policy to cover the state-space. Once the agent observe rewarding behaviour, it must then reduce entropy to converge on task-optimal behaviour.

All experiments run on the CPU of a single machine. The agent interacts with the environment and update its parameters synchronously in a single stream of experience. A *step* is thus comprised of the following operations, in order: (1) given observation, agent takes action, (2) if applicable, agent update its parameters, (3) environment transitions based on action and return new observation. The parameter update step is implemented differently depending on the agent, described below.

B.2 Actor-Critic Experiments

Agent The first agent we evaluate is a simple actor-critic which implements a softmax policy ($\pi_{\mathbf{x}}$) and a critic ($v_{\mathbf{z}}$) using separate feed-forward MLPs. Agent parameter updates are done according to the actor-critic loss in Eq. 5 with the on-policy n-step return target. For a given parameterisation of the agent, we interact with the environment for $N = 16$ steps, collecting all observations, rewards, and actions into a rollout. When the rollout is full, the agent update its parameters under the actor-critic loss with SGD as the optimiser. To isolate the effect of meta-learning, all hyper-parameters except the entropy regularization weight ($\epsilon = \epsilon_{\text{EN}}$) are fixed (Table 2); for each agent, we sweep for the learning rate that yields highest cumulative reward within a 10 million step budget. For the non-adaptive baseline, we additionally sweep for the best regularization weight.

Meta-learning To meta-learn the entropy regularization weight, we introduce a small MLP with meta-parameters $\mathbf{w}$ that ingests a statistic $\mathbf{t}$ of the learning process—the average reward over each of the 10 most recent rollouts—and predicts the entropy rate $\epsilon_{\mathbf{w}}(\mathbf{t}) \in \mathbb{R}_+$ to use in the agent's parameter update of $\mathbf{x}$. To compute meta-updates, for a given horizon $T = K$ or $T = K + (L-1)$, we fix $\mathbf{w}$ and make T agent parameter updates to obtain a sequence $(\tau_1, \mathbf{x}^{(1)}, \mathbf{z}^{(1)}, \ldots, \tau_T, \mathbf{x}^{(T)}, \mathbf{z}^{(T)})$.

MG is optimised by averaging each policy and entropy loss encountered in the sequence, i.e. the meta-objective is given by $\frac{1}{T}\sum_{t=1}^{T} \ell^t_{\text{PG}}(\mathbf{x}^{(t)}(\mathbf{w})) + \epsilon_{\text{meta}}\ell^t_{\text{EN}}(\mathbf{x}^{(t)}(\mathbf{w}))$, where $\epsilon_{\text{meta}} \in \{0, 0.1\}$ is a fixed hyper-parameter and ℓ^t implies that the objective is computed under τ_t.

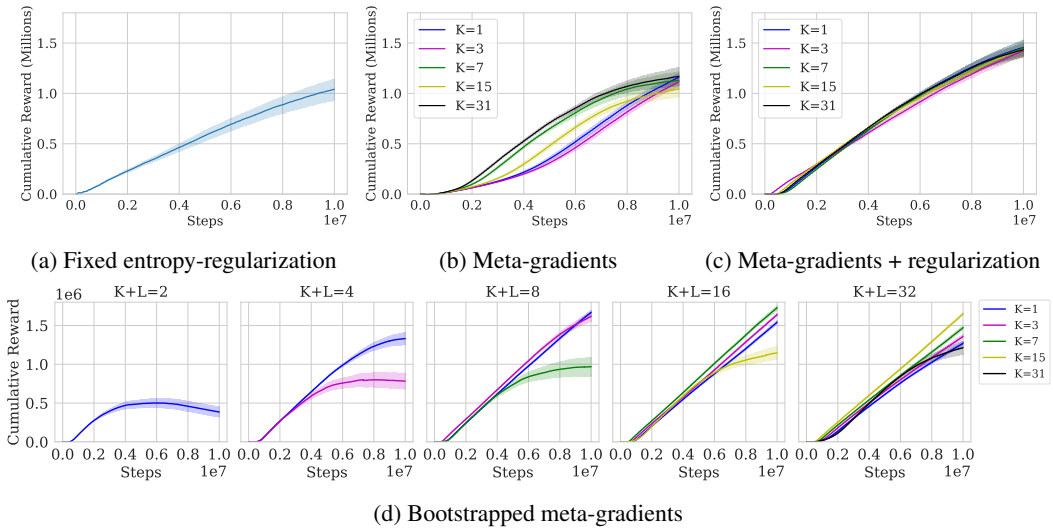

Figure 8: Total rewards on two-colors with actor-critics. Shading: standard deviation over 50 seeds.

BMG is optimised by computing the matching loss $\mu_{\tau_T}(\tilde{\mathbf{x}}, \mathbf{x}^{(K)}(\mathbf{w}))$, where $\tilde{\mathbf{x}}$ is given by $\tilde{\mathbf{x}} = \mathbf{x}^{(T)} - \beta \nabla_x (\ell^T_{\text{PG}}(\mathbf{x}^{(T)}) + \epsilon_{\text{meta}} \ell^T_{\text{EN}}(\mathbf{x}^{(T)}))$. That is to say, the TB "unrolls" the meta-learner for $L-1$ steps, starting from $(\mathbf{x}^{(K)}, \mathbf{z}^{(K)})$, and takes a final policy-gradient step ($\epsilon_{\text{meta}} = 0$ unless otherwise noted). Thus, in this setting, our TB exploits that the first $(L-1)$ steps have already been taken by the agent during the course of learning. Moreover, the final Lth step only differs in the entropy regularization weight, and can therefore be implemented without an extra gradient computation. As such, the meta-update under BMG exhibit no great computational overhead to the MG update. In practice, we observe no significant difference in wall-clock speed for a given K.

Main experiment: detailed results The purpose of our main experiment Section 5.1 is to (a) test whether larger meta-learning horizons—particularly by increasing L—can mitigate the short-horizon bias, and (b) test whether the agent can learn an exploration schedule without explicit domain knowledge in the meta-objective (in the form of entropy regularization). As reported in Section 5.1, we find the answer to be affirmative in both cases. To shed further light on these findings, Figure 8 reports cumulative reward curves for our main experiment in Section 5.1. We note that MG tends to collapse for any K unless the meta-objective is explicitly regularized via ϵ_{meta}. To characterise why MG fail for $\epsilon_{\text{meta}} = 0$, Figure 9 portrays the policy entropy range under either MG or BMG. MG is clearly overly myopic by continually shrinking the entropy range, ultimately resulting in a non-adaptive policy.

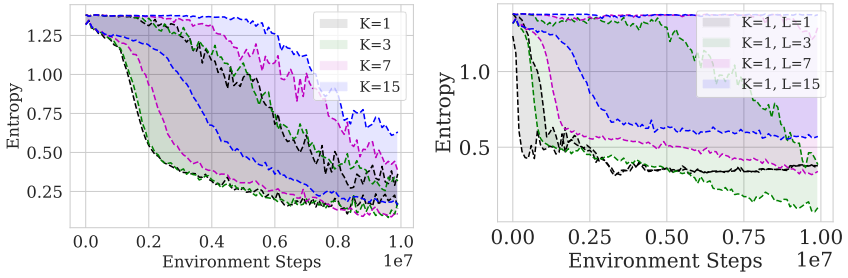

Figure 9: Range of the entropy of a softmax-policy over time (2-colors). Each shaded area shows the difference between the entropy 3333 steps after the agent observes a new entropy and the entropy after training on the reward-function for 100000 steps. Meta-gradients without explicit entropy-regularization (left) reduce entropy over time while Bootstrapped meta-gradients (right) maintain entropy with a large enough meta-learning horizon. Averaged across 50 seeds.

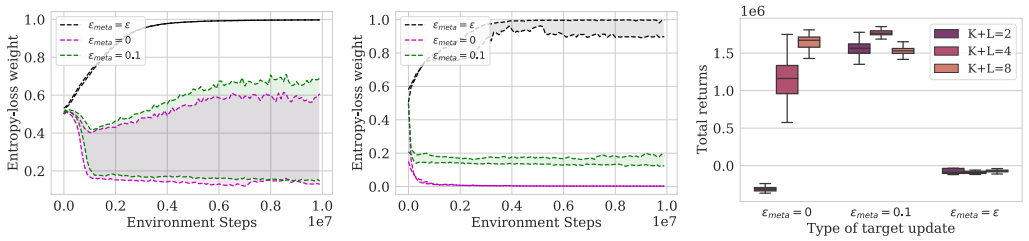

Figure 10: Ablations for actor-critic agent with BMG. Each shaded area shows the range of entropy regularization weights generated by the meta-learner. The range is computed as the difference between ϵ at the beginning and end of each reward-cycle. Center: entropy regularization weight range when $K = 1$ and $L = 7$. Right: entropy regularization weight range when $K = 1$ and $L = 1$. Right: For $K = 1$ effect of increasing L with or without meta-entropy regularization. Result aggregated over 50 seeds.

Ablation: meta-regularization To fully control for the role of meta-regularization, we conduct further experiments by comparing BMG with and without entropy regularization (i.e. ϵ_{meta}) in the Lth target update step. Figure 10 demonstrates that BMG indeed suffers from myopia when $L = 1$, resulting in a collapse of the entropy regularization weight range. However, increasing the meta-learning horizon by setting $L = 7$ obtains a wide entropy regularization weight range. While adding meta-regularization does expand the range somewhat, the difference in total return is not statistically significant (right panel, Figure 10).

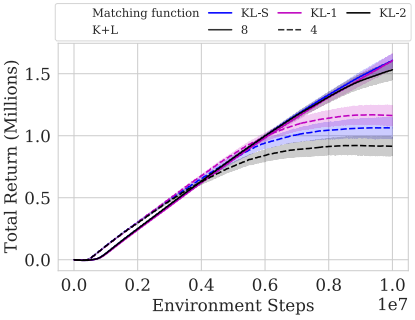

Figure 11: Total reward on two-colors with an actor-critic agent and different matching functions for BMG. Shading: standard deviation over 50 seeds.

Ablation: target bootstrap Our main TB takes $L - 1$ steps under the meta-learned update rule, i.e. the meta-learned entropy regularization weight schedule, and an Lth policy-gradient step without entropy regularization. In this ablation, we very that taking a final step under a *different* update rule is indeed critical. Figure 10 shows that, for $K = 1$ and $L \in \{1, 7\}$, using the meta-learned update rule for all target update steps leads to a positive feedback loop that results in maximal entropy regularization, leading to a catastrophic loss of performance (right panel, Figure 10).

Ablation: matching function Finally, we control for different choices of matching function. Figure 11 contrasts the mode-covering version, KL-1, with the mode-seeking version, KL-2, as well as the symmetric KL. We observe that, in this experiment, this choice is not as significant as in other experiments. However, as in Atari, we find a the mode-covering version to perform slightly better.

B.3 Q-learning Experiments

Agent In this experiment, we test Peng's $Q(\lambda)$ [42] agent with ε-greedy exploration. The agent implements a feed-forward MLP to represent a Q-function $q_{\mathbf{x}}$ that is optimised online. Thus, agent parameter update steps do not use batching but is done online (i.e. on each step). To avoid instability, we use a momentum term that maintains an Exponentially Moving Average (EMA) over the agent parameter gradient. In this experiment we fix all hyper-parameters of the update rule (Table 2) and instead focuses on meta-learned ε-greedy exploration.

BMG We implement BMG in a similar fashion to the actor-critic case. The meta-learner is represented by a smaller MLP $\varepsilon_{\mathbf{w}}(\cdot)$ with meta-parameters $\mathbf{w}$ that ingests the last 50 rewards, denoted by $\mathbf{t}$, and outputs the ε to use on the current time-step. That is to say, given meta-parameters $\mathbf{w}$, the

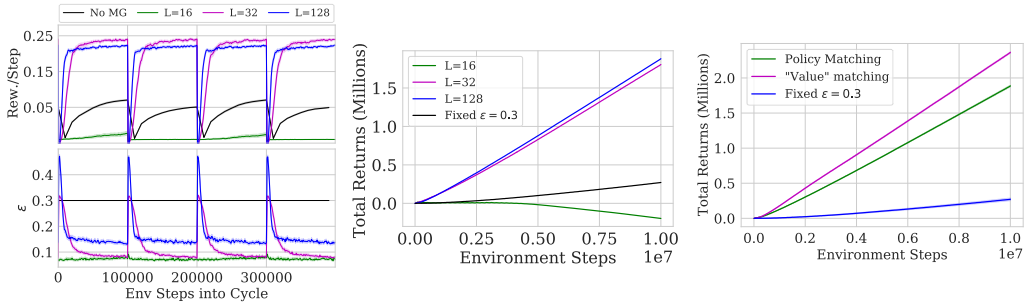

Figure 12: Results on two-colors under a $Q(\lambda)$ agent with meta-learned ε-greedy exploration under BMG. Averaged over 50 seeds.

agent's policy is defined by

$$\pi_{\mathbf{x}}(\mathbf{a} \mid \mathbf{s}_t, \mathbf{t}_t, \mathbf{w}) = \begin{cases} 1 - \varepsilon_{\mathbf{w}}(\mathbf{t}_t) + \frac{\varepsilon_{\mathbf{w}}(\mathbf{t}_t)}{|A|} & \text{if} \quad \mathbf{a} = \arg\max_{\mathbf{b}} q_{\mathbf{x}}(\mathbf{s}_t, \mathbf{b}) \\ \frac{\varepsilon_{\mathbf{w}}(\mathbf{t}_t)}{|A|} & \text{else.} \end{cases}$$

Policy-matching This policy can be seen as a stochastic policy which takes the Q-maximizing action with probability $1 - \varepsilon$ and otherwise picks an action uniformly at random. The level of entropy in this policy is regulated by the meta-learner. We define a TB by defining a target policy under $q_{\tilde{\mathbf{x}}}$, where $\tilde{\mathbf{x}}$ is given by taking L update steps. Since there are no meta-parameters in the update rule, all L steps use the same update rule. However, we define the target policy as the greedy policy

$$\pi_{\tilde{\mathbf{x}}}(\mathbf{a} \mid \mathbf{s}_t) = \begin{cases} 1 & \text{if} \quad \mathbf{a} = \arg\max_{\mathbf{b}} q_{\tilde{\mathbf{x}}}(\mathbf{s}_t, \mathbf{b}) \\ 0 & \text{else.} \end{cases}$$

The resulting BMG update is simple: minimize the KL-divergence $\mu^{\pi}(\tilde{\mathbf{x}}, \mathbf{x}) \coloneqq \mathrm{KL}\left(\pi_{\tilde{\mathbf{x}}} \parallel \pi_{\mathbf{x}}\right)$ by adjusting the entropy in $\pi_{\mathbf{x}}$ through $\varepsilon_{\mathbf{w}}$. Thus, policy-matching under this target encourages the meta-learner to match a greedy policy-improvement operation on a target $q_{\tilde{\mathbf{x}}}$ that has been trained for a further L steps. More specifically, if $\arg\max_{\mathbf{b}} q_{\tilde{\mathbf{x}}}(\mathbf{s}, \mathbf{b}) = \arg\max_{\mathbf{b}} q_{\mathbf{x}}(\mathbf{s}, \mathbf{b})$, so that the greedy policy improvement matches the target, then the matching loss is minimised by setting $\varepsilon = 0$. If greedy policy improvement does not correspond, so that acting greedily w.r.t. $q_{\mathbf{x}}$ does not match the target, then the matching loss is minimised by increasing entropy, i.e. increasing ε. The meta-objective is defined in terms of $\mathbf{x}$ as it does not require differentiation through the update-rule.

'Value'-matching A disadvantage of policy matching is that it provides a sparse learning signal: ε is increased when the target-policy differs from the current policy and decreased otherwise. The magnitude of the change depends solely on the current value of ε. It is therefore desirable to evaluate alternative matching functions that provide a richer signal. Inspired by value-matching for actor-critic agents, we construct a form of 'value' matching by taking the expectation over $q_{\mathbf{x}}$ under the induced stochastic policy, $u_{\mathbf{x}}(\mathbf{s}) \coloneqq \sum_{\mathbf{a} \in \mathcal{A}} \pi_{\mathbf{x}}(\mathbf{a} \mid \mathbf{s}) q_{\mathbf{x}}(\mathbf{s}, \mathbf{a})$. The resulting matching objective is given by

$$\mu^{u}(\tilde{\mathbf{x}}, \mathbf{x}) = \mathbb{E}\left[\left(u_{\tilde{\mathbf{x}}}(\mathbf{s}) - u_{\mathbf{x}}(\mathbf{s}; \mathbf{t}, \mathbf{w})\right)^2\right].$$

While the objective is structurally similar to value-matching, u does not correspond to well-defined value-function since $q_{\mathbf{x}}$ is not an estimate of the action-value of $\pi_{\mathbf{x}}$.

Detailed results Figure 12 shows the learned ε-schedules for different meta-learning horizons: if L is large enough, the agent is able to increase exploration when the task switches and quickly recovers a near-optimal policy for the current cycle. Figure 12 further shows that a richer matching function, in this case in the form of 'value' matching, can yield improved performance.

Table 2: Two-colors hyper-parameters

Actor-critic	
Inner Learner	
Optimiser	SGD
Learning rate	0.1
Batch size	16 (losses are averaged)
γ	0.99
μ	$\mathrm{KL}(\pi_{\tilde{x}}\|\|\pi_{x'})$
MLP hidden layers (v, π)	2
MLP feature size (v, π)	256
Activation Function	ReLU
Meta-learner	
Optimiser	Adam
ϵ (Adam)	10^{-4}
β_1, β_2	0.9, 0.999
Learning rate candidates	$\{3 \cdot 10^{-6}, 10^{-5}, 3 \cdot 10^{-5}, 10^{-4}, 3 \cdot 10^{-4}\}$
MLP hidden layers (ϵ)	1
MLP feature size (ϵ)	32
Activation Function	ReLU
Output Activation	Sigmoid

$Q(\lambda)$	
Inner Learner	
Optimiser	Adam
Learning Rate	$3 \cdot 10^{-5}$
ϵ (Adam)	10^{-4}
β_1, β_2	0.9, 0.999
Gradient EMA	0.9
λ	0.7
γ	0.99
MLP hidden layers (Q)	2
MLP feature size (Q)	256
Activation Function	ReLU
Meta-learner	
Learning Rate	10^{-4}
ϵ (Adam)	10^{-4}
β_1, β_2	0.9, 0.999
Gradient EMA	0.9
MLP hidden layers (ϵ)	1
MLP feature size (ϵ)	32
Activation Function	ReLU
Output Activation	Sigmoid

C Atari

Setup Hyper-parameters are reported in Table 3. We follow the original IMPALA [15] setup, but do not down-sample or gray-scale frames from the environment. Following previous works [66, 68], we treat each game level as a separate learning problem; the agent is randomly initialized at the start of each learning run and meta-learning is conducted online during learning on a single task. We evaluate final performance between 190-200 million frames. All experiments are conducted with 3 independent runs under different seeds. Each of the 57 levels in the Atari suite is a unique environment with distinct visuals and game mechanics. Exploiting this independence, statistical tests of aggregate performance relies on a total sample size per agent of $3 \times 57 = 171$.

Agent We use a standard feed-forward agent that received a stack of the 4 most recent frames [36] and outputs a softmax action probability along with a value prediction. The agent is implemented as a deep neural network; we use the IMPALA network architecture without LSTMs, with larger convolution kernels to compensate for more a complex input space, and with a larger conv-to-linear projection. We add experience replay (as per [49]) to allow multiple steps on the target. All agents use the same number of online samples; unless otherwise stated, they also use the same number of replay samples. We ablate the role of replay data in Appendix C.2.

STACX The IMPALA agent introduces specific form of importance sampling in the actor critic update and while STACX largely rely on the same importance sampling mechanism, it differs slightly to facilitate the meta-gradient flow. The actor-critic update in STACX is defined by Eq. 5 with the following definitions of ρ and G. Let $\bar{\rho} \geq \bar{c} \in \mathbb{R}_+$ be given and let $\nu : \mathcal{S} \times \mathcal{A} \to [0, 1]$ represent the *behaviour policy* that generated the rollout. Given $\pi_{\mathbf{x}}$ and $v_{\bar{\mathbf{z}}}$, define the Leaky V-Trace target by

$$
\begin{aligned}
\eta_t &\coloneqq \pi_{\mathbf{x}}(\mathbf{a}_t \mid \mathbf{s}_t) \,/\, \nu(\mathbf{a}_t \mid \mathbf{s}_t) \\
\rho_t &\coloneqq \alpha_\rho \min\{\eta_t, \bar{\rho}\} + (1 - \alpha_\rho)\eta_t \\
c_i &\coloneqq \lambda \left(\alpha_c \min\{\eta_i, \bar{c}\} + (1 - \alpha_c)\eta_i\right) \\
\delta_t &\coloneqq \rho_t \left(\gamma v_{\bar{\mathbf{z}}}(\mathbf{s}_{t+1}) + r_{t+1} - v_{\bar{\mathbf{z}}}(\mathbf{s}_t)\right) \\
G_t^{(n)} &= v_{\bar{\mathbf{z}}}(\mathbf{s}_t) + \sum_{i=0}^{(n-1)} \gamma^i \left(\prod_{j=0}^{i-1} c_{t+j}\right) \delta_{t+i},
\end{aligned}
$$

with $\alpha_\rho \geq \alpha_c$. Note that—assuming $\bar{c} \geq 1$ and $\lambda = 1$—in the on-policy setting this reduces to the n-step return since $\eta_t = 1$, so $\rho_t = c_t = 1$. The original v-trace target sets $\alpha_\rho = \alpha_c = 1$.

Table 3: Atari hyper-parameters

ALE [8]	
Frame dimensions (H, W, D)	160, 210, 3
Frame pooling	None
Frame grayscaling	None
Num. stacked frames	4
Num. action repeats	4
Sticky actions [33]	False
Reward clipping	$[-1, 1]$
$\gamma = 0$ loss of life	True
Max episode length	108 000 frames
Initial noop actions	30
IMPALA Network [15]	
Convolutional layers	4
Channel depths	$64, 128, 128, 64$
Kernel size	3
Kernel stride	1
Pool size	3
Pool stride	2
Padding	'SAME'
Residual blocks per layer	2
Conv-to-linear feature size	512
STACX [68]	
Auxiliary tasks	2
MLP hidden layers	2
MLP feature size	256
Max entropy loss value	0.9
Optimisation	
Unroll length	20
Batch size	18
of which from replay	*12*
of which is online data	*6*
Replay buffer size	10 000
LASER [49] KL-threshold	2
Optimiser	RMSProp
Initial learning rate	10^{-4}
Learning rate decay interval	200 000 frames
Learning rate decay rate	Linear to 0
Momentum decay	0.99
Epsilon	10^{-4}
Gradient clipping, max norm	0.3
Meta-Optimisation	
γ, λ, $\bar{\rho}$, $\bar{c}$, α	0.995, 1, 1, 1, 1
ϵ_{PG}, ϵ_{EN}, ϵ_{TD}	1, 0.01, 0.25
Optimiser	Adam
Learning rate	10^{-3}
β_1, β_2	0.9, 0.999
Epsilon	10^{-4}
Gradient clipping, max norm	0.3

STACX defines the main "task" as a tuple $(\pi^0, v^0, f(\cdot, \mathbf{w}_0)$, consisting of a policy, critic, and an actor-critic objective (Eq. 5) under Leaky V-trace correction with meta-parameters $\mathbf{w}_0$. Auxiliary tasks are analogously defined tuples $(\pi^i, v^i, f(\cdot, \mathbf{w}_i))$, $i \geq 1$. All policies and critics share the same feature extractor but differ in a separate MLP for each π^i and v^i. The objectives differ in their hyper-parameters, with all hyper-parameters being meta-learned. Auxiliary policies are not used for acting; only the main policy π^0 interacts with the environment. The objective used to update the agent's parameters is the sum of all tasks (each task is weighted through $\epsilon_{\text{PG}}, \epsilon_{\text{EN}}, \epsilon_{\text{TD}}$). The objective used for the MG update is the original IMPALA objective under fixed hyper-parameters $\mathbf{p}$ (see Meta-Optimisation in Table 3). Updates to agent parameters and meta-parameters happen simultaneously on rollouts τ. Concretely, let $\mathbf{m}$ denote parameters of the feature extractor, with $(\mathbf{x}_i, \mathbf{z}_i)$ denoting parameters of task i's policy MLP and critic MLP. Let $\mathbf{u}_i \coloneqq (\mathbf{m}, \mathbf{x}_i, \mathbf{z}_i)$ denote parameters of (π^i, v^i), with $\mathbf{u} \coloneqq (\mathbf{m}, \mathbf{x}_0, \mathbf{z}_0, \dots \mathbf{x}_n, \mathbf{z}_n)$. Let $\mathbf{w} = (\mathbf{w}_0, \dots, \mathbf{w}_n)$ and denote by $\mathbf{h}$ auxiliary vectors of the optimiser. Given (a batch of) rollout(s) τ, the STACX update is given by

$$\left(\mathbf{u}^{(1)}, \mathbf{h}_u^{(1)}\right) = \text{RMSProp}\left(\mathbf{u}, \mathbf{h}_u, \mathbf{g}_u\right) \qquad \mathbf{g}_u = \nabla_u \sum_{i=1}^{n} f_\tau\left(\mathbf{u}_i; \mathbf{w}_i\right)$$

$$\left(\mathbf{w}^{(1)}, \mathbf{h}_w^{(1)}\right) = \text{Adam}\left(\mathbf{w}, \mathbf{h}_w, \mathbf{g}_w\right) \qquad \mathbf{g}_w = \nabla_w f_\tau\left(\mathbf{u}_0^{(1)}(\mathbf{w}); \mathbf{p}\right).$$

BMG We use the same setup, architecture, and hyper-parameters for BMG as for STACX unless otherwise noted; the central difference is the computation of $\mathbf{g}_w$. For $L = 1$, we compute the bootstrapped meta-gradient under μ_τ on data τ by

$$\mathbf{g}_w = \nabla_w \mu_\tau \left(\tilde{\mathbf{u}}_0, \mathbf{u}_0^{(1)}(\mathbf{w})\right), \quad \text{where} \quad \left(\tilde{\mathbf{u}}_0, \_\right) = \text{RMSProp}\left(\mathbf{u}_0^{(1)}, \mathbf{h}_u^{(1)}, \nabla_u f_\tau\left(\mathbf{u}_0^{(1)}; \mathbf{p}\right)\right).$$

Note that the target uses the same gradient $\nabla_u f(\mathbf{u}_0^{(1)}; \mathbf{p})$ as the outer objective in STACX; hence, BMG does not use additional gradient information or additional data for $L = 1$. The *only* extra computation is the element-wise update required to compute $\tilde{\mathbf{u}}_0$ and the computation of the matching loss. We discuss computational considerations in Appendix C.4. For $L > 1$, we take $L - 1$ step under the meta-learned objective with different replay data in each update. To write this explicitly, let τ be the rollout data as above. Let $\tilde{\tau}^{(l)}$ denote a separate sample of only replay data used in the lth target update step. For $L > 1$, the TB is described by the process

$$\left(\tilde{\mathbf{u}}_0^{(1)}, \tilde{\mathbf{h}}_u^{(1)}\right) = \text{RMSProp}\left(\mathbf{u}_0^{(1)}, \mathbf{h}_u^{(1)}, \mathbf{g}_u^{(1)}\right), \qquad \mathbf{g}_u^{(1)} = \nabla_u \sum_{i=1}^{n} f_{\tilde{\tau}^{(1)}}\left(\mathbf{u}_i^{(1)}; \mathbf{w}_i\right)$$

$$\left(\tilde{\mathbf{u}}_0^{(2)}, \tilde{\mathbf{h}}_u^{(2)}\right) = \text{RMSProp}\left(\tilde{\mathbf{u}}_0^{(1)}, \tilde{\mathbf{h}}_u^{(1)}, \tilde{\mathbf{g}}_u^{(1)}\right), \qquad \tilde{\mathbf{g}}_u^{(1)} = \nabla_u \sum_{i=1}^{n} f_{\tilde{\tau}^{(2)}}\left(\tilde{\mathbf{u}}_i^{(1)}; \mathbf{w}_i\right)$$

$$\vdots$$

$$\left(\tilde{\mathbf{u}}_0, \_\right) = \text{RMSProp}\left(\tilde{\mathbf{u}}_0^{(L-1)}, \tilde{\mathbf{h}}_u^{(L-1)}, \tilde{\mathbf{g}}_u^{(L-1)}\right), \qquad \tilde{\mathbf{g}}_u^{(L-1)} = \nabla_u f_\tau\left(\tilde{\mathbf{u}}_0^{(L-1)}, \mathbf{p}\right).$$

Targets and corresponding momentum vectors are discarded upon computing the meta-gradient. This TB corresponds to following the meta-learned update rule for $L - 1$ steps, with a final step under the IMPALA objective. We show in Appendix C.3 that this final step is crucial to stabilise meta-learning.

Matching functions are defined in terms of the rollout τ and with targets defined in terms of the main task $\mathbf{u}_0$. Concretely, we define the following objectives:

$$\mu_\tau^\pi \left(\tilde{\mathbf{u}}_0, \mathbf{u}_0^{(1)}(\mathbf{w})\right) = \text{KL}\left(\pi_{\tilde{\mathbf{u}}_0} \parallel \pi_{\mathbf{u}_0^{(1)}(\mathbf{w})}\right),$$

$$\mu_\tau^v \left(\tilde{\mathbf{u}}_0, \mathbf{u}_0^{(1)}(\mathbf{w})\right) = \mathbb{E}\left[\left(v_{\tilde{\mathbf{u}}_0} - v_{\mathbf{u}_0^{(1)}(\mathbf{w})}\right)^2\right],$$

$$\mu_\tau^{\pi+v} \left(\tilde{\mathbf{u}}_0, \mathbf{u}_0^{(1)}(\mathbf{w})\right) = \mu_\tau^\pi \left(\tilde{\mathbf{u}}_0, \mathbf{u}_0^{(1)}(\mathbf{w})\right) + \lambda \mu_\tau^v \left(\tilde{\mathbf{u}}_0, \mathbf{u}_0^{(1)}(\mathbf{w})\right), \qquad \lambda = 0.25,$$

$$\mu^{L2} \left(\tilde{\mathbf{u}}_0, \mathbf{u}_0^{(1)}(\mathbf{w})\right) = \left\| \tilde{\mathbf{u}}_0 - \mathbf{u}_0^{(1)}(\mathbf{w}) \right\|_2.$$

C.1 BMG Decomposition

In this section, we decompose the BMG agent to understand where observed gains come from. To do so, we begin by noting that—by virtue of Eq. 3—STACX is a special case of BMG under $\mu(\tilde{\mathbf{u}}, \mathbf{u}_0^{(1)}(\mathbf{w})) = \|\tilde{\mathbf{u}} - \mathbf{u}_0^{(1)}(\mathbf{w})\|_2^2$ with $\tilde{\mathbf{u}} = \mathbf{u}_0^{(1)} - \frac{1}{2}\nabla_u f_\tau(\mathbf{u}_0^{(1)}; \mathbf{p})$. That is to say, if the target is generated by a pure SGD step and the matching function is the squared L2 objective. We will refer to this configurations as `SGD, L2`. From this baseline—i.e. STACX—a minimal change is to retain the matching function but use RMSProp to generate the target. We refer t o this configuration as `RMS, L2`. From Corollary 1, we should suspect that correcting for curvature should improve performance. While RMSProp is not a representation of the metric G in the analysis, it is nevertheless providing some form of curvature correction. The matching function can then be used for further corrections.

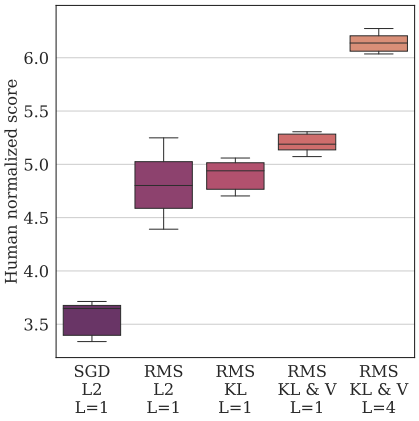

Figure 13: Atari BMG decomposition. We report human normalized score (median, quantiles, $\frac{1}{2}$IQR) between 190-200M frames over all 57 games, with 3 independent runs for each configuration.

Figure 13 shows that changing the target update rule from SGD to RMSProp, thereby correcting for curvature, yields a substantial gain. This supports our main claim that BMG can control for curvature and thereby facilitate meta-optimisation. Using the squared Euclidean distance in parameter space (akin to [38, 17]) is surprisingly effective. However, it exhibits substantial volatility and is prone to crashing (c.f. Figure 15); changing the matching function to policy KL-divergence stabilizes meta-optimisation. Pure policy-matching leaves the role of the critic—i.e. policy evaluation—implicit. Having an accurate value function approximation is important to obtain high-quality policy gradients. It is therefore unsurprising that adding value matching provides a statistically significant improvement. Finally, we find that BMG can also mitigate myopia by extending the meta-learning horizon, in our TB by unrolling the meta-learned update rule for $L - 1$ steps. This is roughly as important as correcting for curvature, in terms of the relative performance gain.

To further support these findings, we estimate the effect BMG has on ill-conditioning and meta-gradient variance on three games where both STACX and BMG exhibit stable learning (to avoid confounding factors of non-stationary dynamics): Kangaroo, Star Gunner, and Ms Pacman. While the Hessian of the meta-gradient is intractable, an immediate effect of ill-conditioning is gradient

Table 4: Meta-gradient cosine similarity and variance per-game at 50-150M frames over 3 seeds.

	KL	KL & V	L2	STACX
Kangaroo				
Cosine similarity	0.19 (0.02)	0.11 (0.01)	0.001 (1e-4)	0.009 (0.01)
Meta-gradient variance	0.05 (0.01)	0.002 (1e-4)	2.3e-9 (4e-9)	6.4e-4 (7e-4)
Meta-gradient norm variance	49	68	47	44
Ms Pacman				
Cosine similarity	0.11 (0.006)	0.03 (0.006)	0.002 (4e-4)	-0.005 (0.01)
Meta-gradient variance	90 (12)	0.8 (0.2)	9.6e-7 (2e-8)	0.9 (0.2)
Meta-gradient norm variance	2.1	7.9	4.2	2.1
Star Gunner				
Cosine similarity	0.13 (0.008)	0.07 (0.001)	0.003 (5e-4)	0.002 (0.02)
Meta-gradient variance	4.2 (1.1)	1.5 (2.3)	1.9e-7 (3e-7)	0.06 (0.03)
Meta-gradient norm variance	6.1	6.6	11.7	6.5

interference, which we can estimate through cosine similarity between consecutive meta-gradients. We estimate meta-gradient variance on a per-batch basis. Table 4 presents mean statistics between 50M and 150M frames, with standard deviation over 3 seeds. BMG achieves a meta-gradient cosine similarity that is generally 2 orders of magnitude larger than that of STACX. It also explicitly demonstrates that using the KL divergence as matching function results in better curvature relative to using the L2 distance. The variance of the meta-gradient is larger for BMG than for STACX (under KL). This is due to intrinsically different gradient magnitudes. To make comparisons, we report the gradient norm to gradient variance ratio, which roughly indicates signal to noise. We note that in this metric, BMG tends to be on par with or lower than that of STACX.

C.2 Effect of Replay

We find that extending the meta-learning horizon by taking more steps on the target leads to large performance improvements. To obtain these improvements, we find that it is critical to re-sample replay data for each step, as opposed to re-using the same data for each rollout. Figure 14 demonstrates this for $L = 4$ on MsPacman. This can be explained by noting that reusing data allows the target to overfit to the current batch. By re-sampling replay data we obtain a more faithful simulation of what the meta-learned update rule would produce in $L - 1$ steps.

The amount of replay data is a confounding factor in the *meta-objective*. We stress that the agent parameter update is *always* the same in any experiment we run. That is to say, the additional use of replay data *only* affects the computation of the meta-objective. To control for this additional data in the meta-objective, we consider a subset of games where we see large improvements from $L > 1$. We run STACX and BMG with $L = 1$, but increase the amount of replay data used to compute the *meta-objective* to match the total amount of replay data used in the meta-objective when $L = 4$. This changes the online-to-replay ratio from $6 : 12$ to $6 : 48$ in the meta objective.

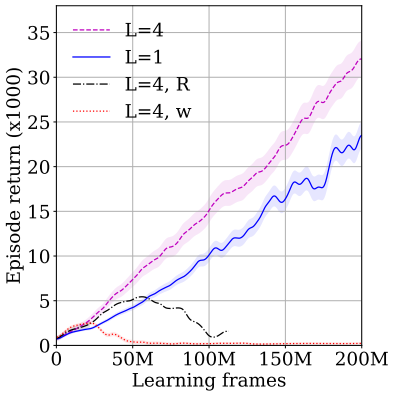

Figure 14: Atari, learning curves on MS Pacman for KL&V. $L = 4, R$ computes the $L$the step on only replay data. $L = 4, w$ uses the meta-learned objective for the Lth step (with Lth step computed on online and replay data, as per default). Shading depicts standard deviation across 3 seeds.

Figure 15 shows that the additional replay data is not responsible for the performance improvements we see for $L = 4$. In fact, we find that increasing the amount of replay data in the meta-objective exacerbates off-policy issues and leads to *reduced* performance. It is striking that BMG can make use of this extra off-policy data. Recall that we use *only* off-policy replay data to take the first $L - 1$ steps on the target, and use the original online-to-replay ratio ($6 : 12$) in the Lth step. In Figure 14, we test the effect of using *only* replay for all L steps and find that having online data in the Lth update step is critical. These results indicate that BMG can make effective use of replay by simulating the effect of the meta-learned update rule on off-policy data and correct for potential bias using online data.

C.3 L vs K

Given that increasing L yields substantial gains in performance, it is interesting to compare against increasing K, the number of agent parameter updates to backpropagate through. For fair comparison, we use an identical setup as for $L > 1$, in the sense that we use new replay data for each of the initial $K - 1$ steps, while we use the default rollout τ for the Kth step. Hence, the data characteristics for $K > 1$ are identical to those of $L > 1$.

However, an important difference arise because each update step takes K steps on the agent's parameters. This means that—withing the 200 million frames budget, $K > 1$ has a computational advantage as it is able to do more updates to the agent's parameters. With that said, these additional $K - 1$ updates use replay data only.

Figure 16 demonstrates that increasing K is fundamentally different from increasing L. We generally observe a loss of performance, again due to interference from replay. This suggests that target bootstrapping allows a fundamentally different way of extending the meta-learning horizon. In

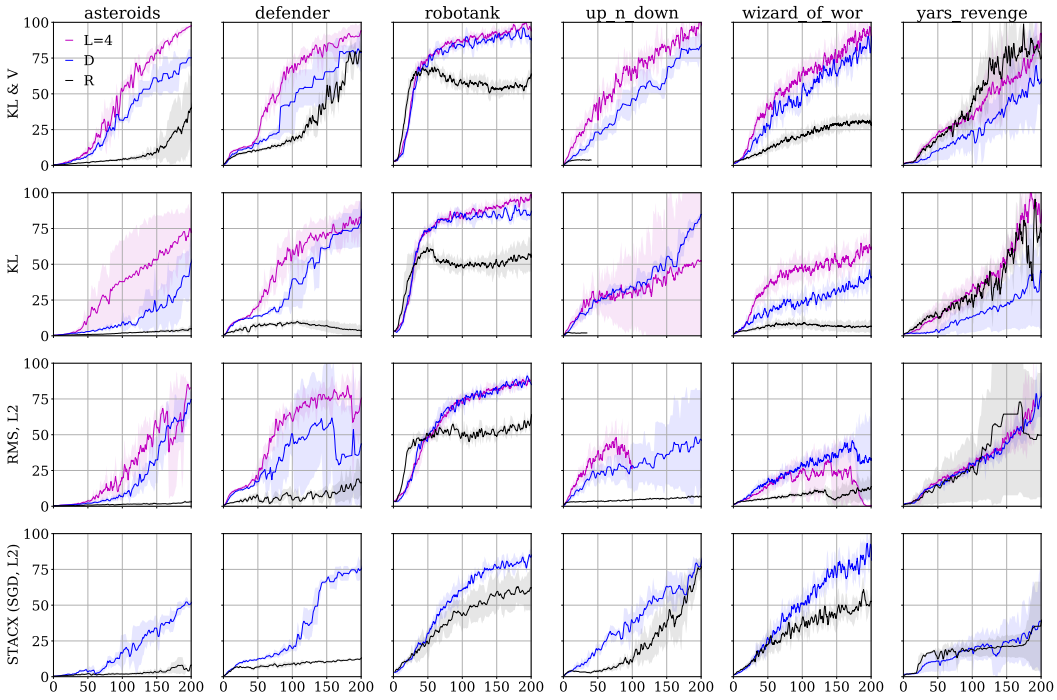

Figure 15: Atari experience replay ablation. We report episode returns, normalized to be in the range $[0, \text{max return}]$ for each game for ease of comparison. Shading depicts standard deviation across 3 seeds. D denotes default BMG configuration for $L = 1$, with $L = 4$ analgously defined. R denotes $L = 1$, but with additional replay in the meta-objective to match the amount of replay used in $L = 4$.

particular, these results suggests that meta-bootstrapping allows us to use relatively poor-quality (as evidence by $K > 1$) approximations to long-term consequences of the meta-learned update rule without impairing the agent's actual parameter update. Finally, there are substantial computational gains from increasing the meta-learning horizon via L rather than K (Figure 17).

C.4 Computational characteristics

IMPALA's distributed setup is implemented on a single machine with 56 CPU cores and 8 TPU [27] cores. 2 TPU cores are used to act in 48 environments asynchronously in parallel, sending rollouts to a replay buffer that a centralized learner use to update agent parameters and meta-parameters. Gradient computations are distributed along the batch dimension across the remaining 6 TPU cores. All Atari experiments use this setup; training for 200 millions frames takes 24 hours.

Figure 17 describes the computational properties of STACX and BMG as a function of the number of agent parameters and the meta-learning horizon, H. For STACX, the meta-learning horizon is defined by the number of update steps to backpropagate through, K. For BMG, we test one version which holds $L = 1$ fixed and varies K, as in for STACX, and one version which holds $K = 1$ ficed and varies L. To control for network size, we vary the number of channels in the convolutions of the network. We use a base of channels per layer, $x = (16, 32, 32, 16)$, that we multiply by a factor $1, 2, 4$. Thus we consider networks with kernel channels $1x = (16, 32, 32, 16)$, $2x = (32, 64, 64, 32)$, and $4x = (64, 128, 128, 64)$. Our main agent uses a network size (Table 3) equal to $4x$. We found that larger networks would not fit into memory when $K > 1$.

First, consider the effect of increasing K (with $L = 1$ for BMG). For the small network ($1x$), BMG is roughly on par with STACX for all values of K considered. However, BMG exhibits poorer scaling in network size, owing to the additional update step required to compute the target bootstrap. For $4x$, our main network configuration, we find that BMG is 20% slower in terms of wall-clock time. Further, we find that neither STACX nor BMG can fit the $4x$ network size in memory when $K = 8$.

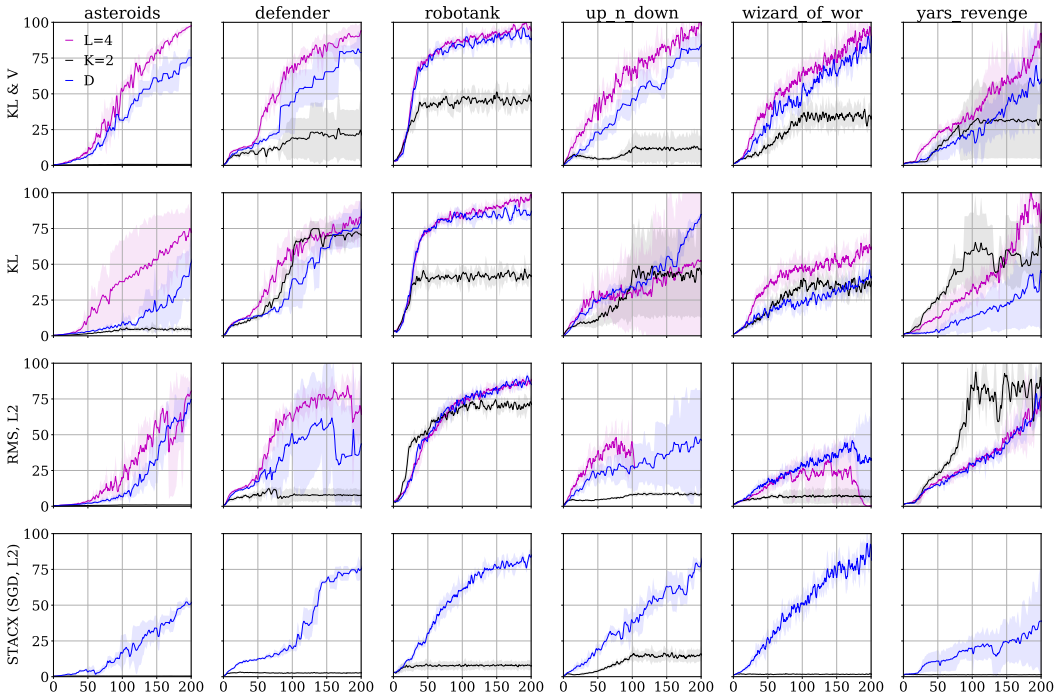

Figure 16: Atari K vs L ablation. We report episode returns, normalized to be in the range $[0, \text{max return}]$ for each game for ease of comparison. Shading depicts standard deviation across 3 seeds. D denotes default BMG configuration for $L = 1$, with $L = 4$ analogously defined. $K = 2$ denotes $L = 1$, but $K = 2$ steps on agent parameters.

Second, consider the effect of increasing L with BMG (with $K = 1$). For $1x$, we observe no difference in speed for any H. However, increasing L exhibits a dramatic improvement in scaling for $H > 2$—especially for larger networks. In fact, $L = 4$ exhibits a factor 2 speed-up compared to STACX for $H = 4, 4x$ and is two orders of magnitude faster for $H = 8, 2x$.

C.5 Additional results

Figure 19 presents per-game results learning curve for main configurations considered in this paper. Table 7 presents mean episode returns per game between 190-200 millions frames for all main configurations. Finally, we consider two variations of BMG in the $L = 1$ regime (Figure 18); one version (NS) re-computes the agent update after updating meta-parameters in a form of trust-region method. The other version (DB) exploits that the target has a taken a further update step and uses the target as new agent parameters. While NS is largely on par, interestingly, DB fails completely.

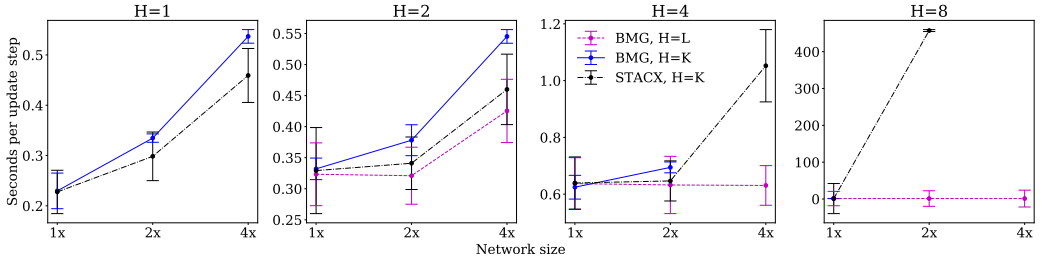

Figure 17: Atari: Computational characteristics as a function of network size (see Appendix C.4) and meta-learning horizon H. When $H = K$, we vary the number of update steps to backpropagate through (with $L = 1$ for BMG). When $H = L$, we vary the number of target update steps (with $K = 1$). Measurements are taken over the first 20 million learning frames on the game Pong.

C.6 Data and hyper-parameter selection

We use the ALE Atari environment, publicly available at https://github.com/mgbellemare/Arcade-Learning-Environment, licensed under GNU GPL 2.0. Environment hyper-parameters were selected based on prior works [36, 15, 68, 49]. Network, optimisation and meta-optimisation hyper-parameters are based on the original STACX implementation and tuned for optimal performance. Our median human normalized score matches published results. For BMG, we did not tune these hyper-parameters, except for $L > 1$. In this case, we observed that unique replay data in the initial $L - 1$ steps was necessary to yield any benefits. We observed a tendency to crash, and thus reduced the gradient clipping ratio from .3 to .2. For BMG configurations that use both policy and value matching, we tuned the weight on value matching by a grid search over $\{0.25, 0.5, 0.75\}$ on Ms Pacman, Zaxxon, Wizard of Wor, and Seaquest, with 0.25 performing best.

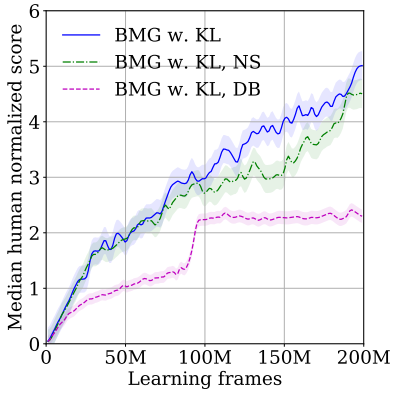

Figure 18: Atari BMG, alternative meta-update strategies. NS re-computes the agent-update the meta-update, akin to a trust-region method. DB uses the bootstrap target as the next agent parameters. Shading depicts standard deviation across 3 seeds.

D Multi-task Meta-Learning

D.1 Problem Formulation

Let $p(\tau)$ denote a given task distribution, where $\tau \in \mathbb{N}$ indexes a task f^τ. Each task is also associated with distinct learner states $\mathbf{h}_\tau$ and task parameters $\mathbf{x}_\tau$, but all task learners use the same meta-learned update rule defined by meta-parameters $\mathbf{w}$. Hence, the meta-learner's problem is again to learn an update rule, but now in expectation over all learning problems. The MG update (Eq. 1) thus takes the form $\mathbf{w}' = \mathbf{w} - \beta \nabla_w \mathbb{E}_\tau[f^\tau(\mathbf{x}_\tau^{(K)}(\mathbf{w}))]$, where the expectation is with respect to $(f^\tau, \mathbf{h}_\tau, \mathbf{x}_\tau)$ and $\mathbf{x}_\tau^{(K)}(\mathbf{w})$ is the K-step update on task τ given $(f^\tau, \mathbf{h}_\tau, \mathbf{x}_\tau)$. Since $p(\tau)$ is independent of $\mathbf{w}$, this update becomes $\mathbf{w}' = \mathbf{w} - \beta \mathbb{E}_\tau[\nabla_w f^\tau(\mathbf{x}_\tau^{(K)}(\mathbf{w}))]$, i.e. the single-task meta-gradient in Section 3 in expectation over the task distribution.

With that said, the expectation involves integrating over $(\mathbf{h}_\tau, \mathbf{x}_\tau)$. This distribution is defined differently depending on the problem setup. In few-shot learning, $\mathbf{x}_\tau$ and $\mathbf{h}_\tau$ are typically a shared initialisations [16, 38, 17] and f^τ differ in terms of the data [61]. However, it is possible to view the expectation as a prior distribution over task parameters [19, 18]. In online multi-task learning, this expectation often reduces to an expectation over current task-learning states [45, 13].

The BMG update is analogously defined. Given a TB ξ, define the task-specific target $\tilde{\mathbf{x}}_\tau$ given $\mathbf{x}_\tau^{(K)}$ by $\xi(\mathbf{x}_\tau^{(K)})$. The BMG meta-loss takes the form $\mathbf{w}' = \mathbf{w} - \beta \nabla_w \mathbb{E}_\tau[\mu_\tau(\tilde{\mathbf{x}}_\tau, \mathbf{x}_\tau^{(K)}(\mathbf{w}))]$, where μ_τ is defined on data from task τ. As with the MG update, as the task distribution is independent of $\mathbf{w}$, this simplifies to $\mathbf{w}' = \mathbf{w} - \beta \mathbb{E}_\tau[\nabla_w \mu_\tau(\tilde{\mathbf{x}}_\tau, \mathbf{x}_\tau^{(K)}(\mathbf{w}))]$, where μ_τ is the matching loss defined on task data from τ. Hence, as with MG, the multi-task BMG update is an expectation over the single-task BMG update in Section 3.

D.2 Few-Shot MiniImagenet

Setup MiniImagenet [61, 43] is a sub-sample of the Imagenet dataset [14]. Specifically, it is a subset of 100 classes sampled randomly from the 1000 classes in the ILSVRC-12 training set, with 600 images for each class. We follow the standard protocol [43] and split classes into a non-overlapping meta-training, meta-validation, and meta-tests sets with 64, 16, and 20 classes in each, respectively. The datasset is licenced under the MIT licence and the ILSVRC licence. The dataset can be obtained from https://paperswithcode.com/dataset/miniimagenet-1. M-shot-N-way classification tasks are sampled following standard protocol [61]. For each task, M classes are randomly sampled from the train, validation, or test set, respectively. For each class, K observations are randomly sampled without replacement. The task validation set is constructed similarly from a

Table 5: Effect of BMG on ill-conditioning and meta-gradient variance on 5-way-5-shot MiniImagenet. Estimated meta-gradient cosine similarity (θ) between consecutive gradients, meta-gradient variance ($\mathbb{V}$), and meta-gradient norm to variance ratio (ρ). Standard deviation across 5 independent seeds.

		MAML			BMG		
K	L	θ	$\mathbb{V}$	ρ	θ	$\mathbb{V}$	ρ
1	1	0.17 (0.01)	0.21 (0.01)	0.02 (0.02)	0.17 (0.01)	0.0002 (5e-6)	0.59 (0.03)
	5				0.18 (0.01)	0.001 (1e-5)	0.23 (0.01)
	10				0.19 (0.01)	0.0003 (2e-5)	0.36 (0.01)
5	1	0.03 (0.01)	0.07 (0.009)	0.08 (0.03)	0.03 (0.005)	0.01 (9e-5)	0.84 (0.03
	5				0.04 (0.005)	0.001 (5e-5)	0.46 (0.02)
	10				0.05 (0.004)	0.003 (3e-5)	0.18 (0.02)

disjoint set of $L = 5$ images per class. We follow the original MAML protocol for meta-training [16], taking 5 task adaptation steps during meta-training and 10 adaptation steps during meta testing.

Model and hyper-parameters We use the standard convolutional model [61], which is a 4-layer convolutional model followed by a final linear layer. Each convolutional layer is defined by a 3×3 kernel with 32 channels, strides of 1, with batch normalisation, a ReLU activation and 2×2 max-pooling. We the same hyper-parameters of optimisation and meta-optimisation as in the original MAML implementation unless otherwise noted.

D.3 Analysis

Next, we conduct a similar analysis as on Atari (Appendix C.1) to study the effect BMG has on ill-conditioning and meta-gradient variance. We estimate ill-conditioning through cosine similarity between consecutive meta-gradients, and meta-gradient variance on a per meta-batch basis. We report mean statistics for the 5-way-5-shot setup between 100 000 and 150 000 meta-gradient steps, with standard deviation over 5 independent seeds, in Table 5.

Unsurprisingly, MAML and BMG are similar in terms of curvature, as both can have a KL-divergence type of meta-objective. BMG obtains greater cosine similarity as L increases, suggesting that BMG can transfer more information by having a higher temperature in its target. However, BMG exhibits substantially lower meta-gradient variance, and the ratio of meta-gradient norm to variance is an order of magnitude larger.

D.4 Computational characteristics

Each model is trained on a single machine and runs on a V100 NVIDIA GPU. Table 6 reports the wall-clock time per meta-training step for MAML and BMG, for various values of K and L, in the 5-way-5-shot MiniImagenet experiment.

In our main configuration, i.e. $K = 5, L = 10$, BMG achieves a throughput of 2.5 meta-training steps per second, compared to 4.4 for MAML, making BMG 50% slower. In this setting, BMG has an effective meta-learning horizon of 15, whereas MAML has a horizon of 5. For MAML to achieve an effective horizon of 15, it's throughput would be reduced to 1.4, instead making MAML 56% slower than BMG.

Table 6: Meta-training steps per second for MAML and BMG on 5-way-5-shot MiniImagenet. Standard deviation across 5 seeds in parenthesis.

K	L	$H = K + L$	MAML	BMG
1	1	2	14.3 (0.4)	12.4 (0.5)
	5	6	-	6.9 (0.3)
	10	11	-	4.4 (0.1)
5	1	6	4.4 (0.06)	4.2 (0.04)
	5	10	-	3.2 (0.03)
	10	15	-	2.5 (0.01)
10	1	11	2.3 (0.01)	2.2 (0.01)
	5	15	-	1.9 (0.01)
	10	20	-	1.7 (0.01)
15	-	15	1.4 (0.01)	-
20	-	20	1.1 (0.01)	-

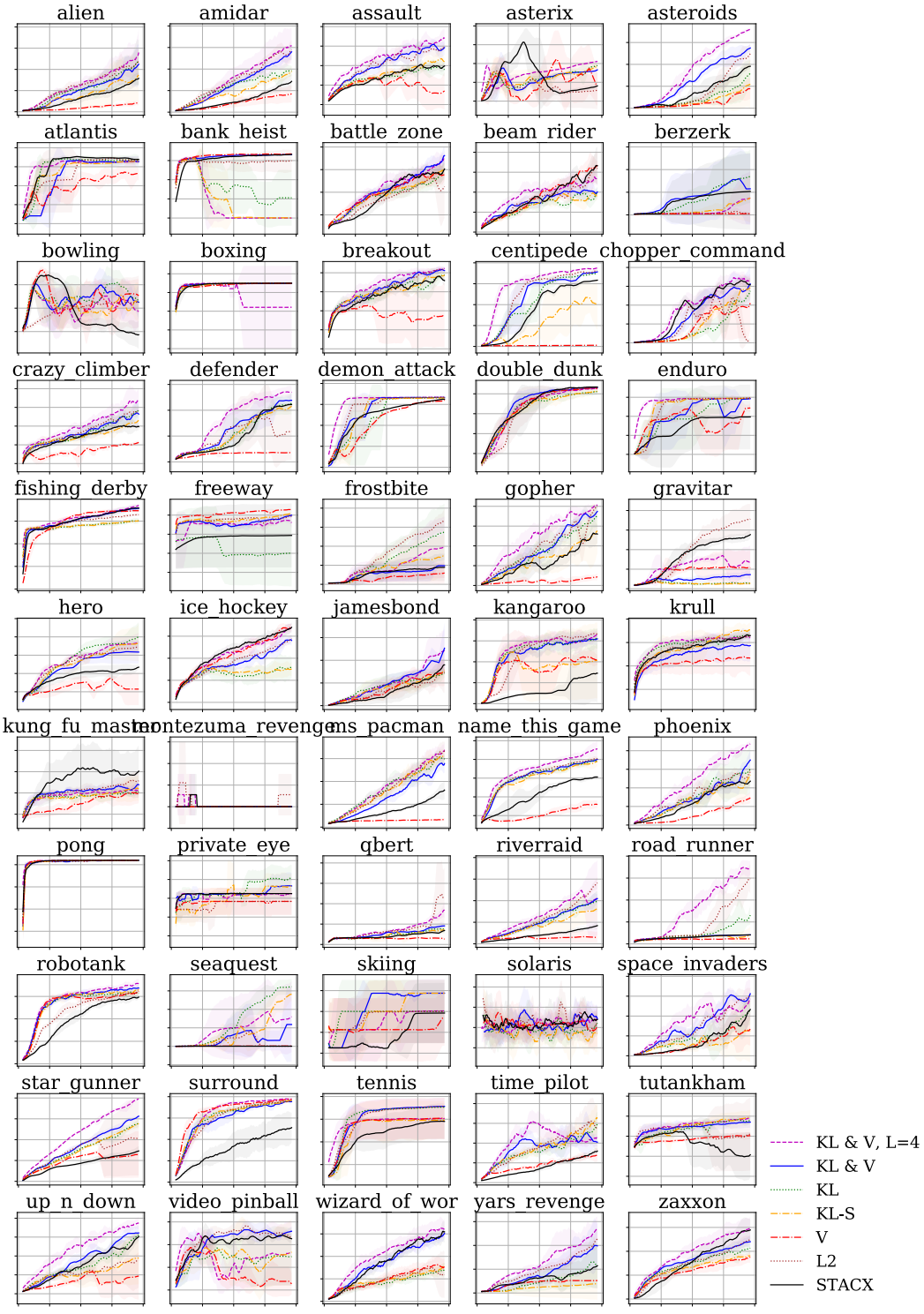

Figure 19: Atari, per-game performance across 3 seeds. Shading depicts standard deviation.

Table 7: Mean per-game performance between 190-200M frames.

	KL	KL & V	KL & V, L=4	KL-S	L2	STACX	V
Alien	45677	44880	58067	35750	50692	31809	7964
Amidar	4800	7099	7528	4974	7691	3719	1896
Assault	20334	29473	33019	21747	28301	19648	4101
Asterix	511550	439475	533385	487367	6798	245617	86053
Asteroids	145337	238320	289689	8585	220366	156096	56577
Atlantis	831920	813772	814780	806698	854441	848007	648988
Bank Heist	571	1325	0	13	1165	1329	1339
Battle Zone	73323	88407	88350	78941	50453	78359	72787
Beam Rider	37170	51649	57409	41454	67726	62892	74397
Berzerk	21146	2946	1588	2183	240	1523	1069
Bowling	46	50	42	46	50	28	52
Boxing	100	100	86	100	100	100	100
Breakout	742	832	847	774	827	717	16
Centipede	537032	542730	558849	291569	550394	478347	8895
Chopper Command	830772	934863	838090	736012	11274	846788	341350
Crazy Climber	233445	212229	265729	199150	229496	182617	126353
Defender	393457	374012	421894	364053	69193	344453	55152
Demon Attack	132508	133109	133571	132529	133469	130741	129863
Double Dunk	22	23	23	21	23	24	23
Enduro	2349	2349	2350	2360	2365	259	2187
Fishing Derby	41	63	68	41	52	62	59
Freeway	10	30	25	31	30	18	33
Frostbite	8820	3895	3995	5547	13477	2522	1669
Gopher	116010	116037	122459	92185	122790	87094	11920
Gravitar	271	709	748	259	3594	2746	944
Hero	60896	48551	52432	56044	51631	35559	20235
Ice Hockey	5	15	20	4	15	19	20
Jamesbond	22129	25951	30157	25766	18200	26123	23263
Kangaroo	12200	12557	13174	1940	13235	3182	8722
Krull	10750	9768	10510	11156	10502	10480	8899
Kung Fu Master	51038	58732	54354	54559	63632	67823	54584
Montezuma Revenge	0	0	0	0	0	0	0
Ms Pacman	25926	22876	28279	26267	27564	12647	2759
Name This Game	31203	31863	36838	30912	32344	24616	12583
Phoenix	529404	542998	658082	407520	440821	370270	247854
Pitfall	0	-1	0	0	0	0	0
Pong	21	21	21	21	21	21	21
Private Eye	165	144	98	130	67	100	68
Qbert	87214	37135	72320	30047	75197	27264	3901
Riverraid	129515	132751	32300	91267	177127	47671	26418
Road Runner	240377	61710	521596	17002	424588	62191	34773
Robotank	64	66	71	65	64	61	65
Seaquest	684870	2189	82925	616738	1477	1744	3653
Skiing	-10023	-8988	-9797	-8988	-9893	-10504	-13312
Solaris	2120	2182	2188	1858	2194	2326	2202
Space Invaders	35762	54046	40790	11314	49333	34875	15424
Star Gunner	588377	663477	790833	587411	39510	298448	43561
Surround	9	9	10	9	9	3	9
Tennis	23	24	23	21	24	19	24
Time Pilot	94746	60918	68626	95854	93466	49932	40127
Tutankham	282	268	291	280	288	101	205
Up N Down	342121	303741	381780	109392	202715	315588	17252
Venture	0	0	0	0	0	0	0
Video Pinball	230252	479861	399094	505212	485852	441220	77100
Wizard Of Wor	21597	45731	49806	22936	10817	47854	24250
Yars Revenge	77001	286734	408061	32031	398656	113651	77169
Zaxxon	44280	49448	59011	36261	49734	56952	35494