Debiasing Meta-Gradient Reinforcement Learning by Learning the Outer Value Function

Anonymous Author(s)
Affiliation
Address
email

Abstract

Meta-gradient Reinforcement Learning (RL) allows agents to self-tune their hyper-parameters in an online fashion during training. In this paper, we identify a bias in the meta-gradient of current meta-gradient RL approaches. This bias comes from using the critic that is trained using the meta-learned discount factor for the advantage estimation in the outer objective which requires a different discount factor. Because the meta-learned discount factor is typically lower than the one used in the outer objective, the resulting bias may cause the meta-gradient to favor myopic policies. We propose a simple solution to this issue: we alleviate this bias by using an alternative, *outer* value function in the estimation of the outer loss. To obtain this outer value function we add a second head to the critic network and train it alongside the classic critic, using the outer loss discount factor. On an illustrative toy problem, we show that the bias can cause catastrophic failure of current meta-gradient RL approaches, and show that our proposed solution fixes it. We then apply our method to more complex environments and demonstrate that fixing the meta-gradient bias significantly improves performance.

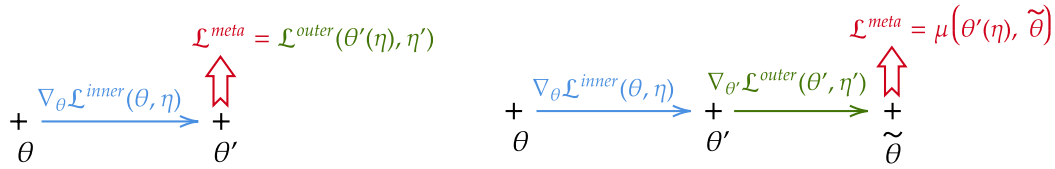
1 Introduction

Recently, reinforcement learning (RL) methods have embraced ideas from meta-learning either to train an agent to quickly adapt to new tasks [7, 13] or to improve the learning process online within a single task [28, 30, 24, 27, 29]. RL algorithms are known to be highly sensitive to their hyperparameters such as the discount factor [1]. In fact, self-tuning these hyper-parameters online has yielded state-of-the-art performance for model-free RL [8]. Meta-gradient RL is the backbone underlying self-tuning and relies on an *inner* loss function used to update the agent's parameters, as well as an *outer* loss to evaluate (or ground) the updated parameters in order to compute a meta-gradient. This meta-gradient is then used to update the meta-parameters (e.g. the discount factor).

In this paper, we focus on the outer loss used by current meta-RL algorithms and show that its estimation is biased. The meta-gradients obtained using this biased estimate can lead to a failure of the self-tuning, potentially resulting in a myopic policy. We propose a computationally cheap approach to eliminate the meta-gradient bias and observe that it solves catastrophic failure in a toy environment and improves performance on a more challenging deep RL task.

There have been several works focusing on bias and variance in meta-gradients. (Bonnet et al., 2021) [6] focuses on multi-step meta-gradients and studies a different kind of bias, namely one with respect to an oracle self-tuning objective, whereas we consider only meta-gradients through one inner

Submitted to the 6th Workshop on Meta-Learning at NeurIPS 2022, New Orleans. Do not distribute.



(a) Meta-gradients (MG)

(b) Bootstrapped meta-gradients (BMG)

Figure 1: The two paradigms of meta-gradients for self-tuning. First, parameters θ are updated using an inner loss and meta-parameters η. Then, a meta-loss is computed from the updated θ'. Meta-gradients (MG) and bootstrapped meta-gradients (BMG) differ in their choice of meta-loss.

update in this work. (Vuorio et al., 2022) [25] studies a different bias, namely the sampling bias, and (Liu et al., 2021) [16] addresses a compositional bias. Although complementary to biases analyzed in these works, the bias we highlight in this paper is different and specific to self-tuning RL.

2 Background

Reinforcement Learning: The goal of an RL agent is to learn the policy π that maximizes the γ-discounted expected return: $\mathbb{E}_{\pi|s_t=s}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right]$, where r_t and s_t are the reward and state at timestep t. Actor-critic algorithms [14, 18] learn a parameterized value function (the critic) and a parameterized policy which is updated in a direction informed by the critic. More precisely, the critic learns the value function $V_{\theta_c}^{\pi,\gamma} = \mathbb{E}_{\pi|s_t=s}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right]$ with critic parameters θ_c, and policy π_{θ_p} with policy parameters θ_p. The policy is trained using the policy gradient [26, 21], $\hat{g}_{\text{pg}}$, estimated by Monte Carlo using

$$\hat{g}_{\text{pg}}(\tau) = \left(r + \gamma V_{\theta_c}^{\pi,\gamma}(s')\right) \nabla_{\theta_p} \log \pi_{\theta_p}(a|s) \tag{1}$$

where τ are transitions. Although one could use the advantage [21] instead of the action value function in the estimate to reduce variance [9, 19], we stick to the action value function for simplicity. We denote $\mathcal{L}^{\text{inner}}(\theta, \eta)$, the total loss used in the full actor-critic update, that combines the policy gradient with entropy regularization [22, 18, 11, 10] and a critic update using the mean squared error of the TD error [23]. We provide further discussion and definitions in Appendix A.

Self-Tuning RL: The inner update using the inner loss $\mathcal{L}^{\text{inner}}(\theta, \eta)$ is a differentiable function of the hyper-parameters η (we consider $\eta = \{\gamma\}$ in this work). Self-tuning meta-RL purposely exposes η as trainable meta-parameters and adapts them online during training by computing meta-gradients. The meta-gradient $\nabla_\eta \mathcal{L}^{\text{meta}}$ is defined as the gradient of a meta-loss with respect to the meta-parameters. It is computed in two phases (see figure 1): **(1)** The agent takes one (or multiple) parameter update(s) by running the actor-critic algorithm with meta-parameters η to obtain updated parameters $\theta'(\eta)$. **(2)** The meta-loss is computed from the updated parameters $\theta'(\eta)$ and the meta-gradient is derived $\nabla_\eta \mathcal{L}^{\text{meta}}(\theta'(\eta))$. One can differentiate two kinds of meta-loss (see figure 1). Firstly, *meta-gradients* (MG) [28, 27, 29, 24] compute the gradient of an outer loss $\mathcal{L}^{\text{outer}}$ applied to updated parameters $\theta'(\eta)$. Secondly, *bootstrapped meta-gradients* (BMG) [17, 8] further update the parameters by taking one (or several) gradient step(s) to obtain a target $\tilde{\theta}$ from which to compute a matching loss between $\theta'(\eta)$ and $\tilde{\theta}$. The target updates proposed by [8] can be done with respect to the inner loss $\mathcal{L}^{\text{inner}}$ for all but the last step, which is computed with an outer loss $\mathcal{L}^{\text{outer}}$. Both methods of computing meta-gradients necessitate an outer loss, $\mathcal{L}^{\text{outer}}$, that is defined to be an actor-critic loss similar to $\mathcal{L}^{\text{inner}}$ but uses different hyper-parameters η' than the inner loss. In particular, the policy gradient term of the outer loss uses a different discount factor γ', leading to the following policy gradient estimate $\hat{g}_{\text{pg}}$.

$$\hat{g}'_{\text{pg}}(\tau) = \left(r + \gamma' V_{\theta'}^{\pi,\gamma'}(s')\right) \nabla_{\theta'} \log \pi_{\theta'}(a|s) \tag{2}$$

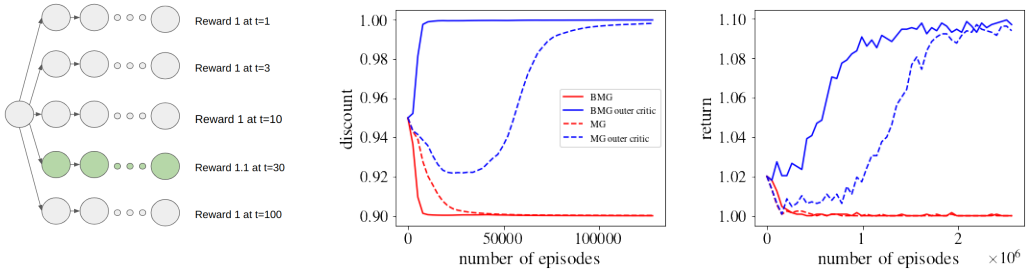


Figure 2: Discounting chain experiment. **(left)** A visualization of the environment. **(middle)** Discount during training. **(right)** Return during training (shares the legend with (middle)). Red curves show current self-tuning methods (MG and BMG) while blue curves are obtained by using an outer-critic to estimate the outer loss.

3 Solution to the meta-gradient bias

In this section, we highlight a bias in the meta-gradient estimation of current meta-RL methods and propose a simple and computationally-efficient way to solve it.

Bias in the meta-gradient: As seen in section 2, the policy gradient terms from the inner and outer loss differ in terms of what hyper-parameters they use. While the inner loss necessitates estimating the value function $V^{\pi,\gamma}$, the outer loss requires $V^{\pi,\gamma'}$. Since the critic is trained to estimate the value function $V^{\pi,\gamma}$, a bias consequently arises when using the critic to estimate the advantage in the policy gradient of the outer loss:

$$\hat{g}'_{\text{pg}}(\tau) = \left(r + \gamma' V^{\pi,\gamma'}(s')\right) \log \pi_{\theta'}(a|s) \not\approx \left(r + \gamma' V^{\pi,\gamma}(s')\right) \log \pi_{\theta'}(a|s) \tag{3}$$

The right-hand side of equation 3 is the biased estimate of the outer policy gradient used by current self-tuning methods.

Removing the bias with an outer-critic head: We propose to learn another critic that we call the *outer-critic*, whose goal is to estimate the *outer* value function, i.e. the value of the policy with (constant) discount γ'. More precisely, the outer-critic learns the value function $V^{\pi,\gamma'}_{\theta_{\bar{c}}}$, parameterized by $\theta_{\bar{c}}$: $V^{\pi,\gamma'}_{\theta_{\bar{c}}}(s) = \mathbb{E}_{\pi|s_t=s}\left[\sum_{k=0}^{\infty} \gamma'^k r_{t+k}\right]$. This way, one can use the outer-critic to produce an unbiased estimate of the outer-loss policy gradient: $\left(r + \gamma' V^{\pi,\gamma'}_{\theta_{\bar{c}}}(s')\right) \nabla_{\theta'_p} \log \pi_{\theta'_p}(a|s)$. To implement the outer-critic, we modify current critic architectures by adding a second head whose goal is to learn the outer value function $V^{\pi,\gamma'}$. As a consequence, our way of removing the bias in the outer loss advantage estimation is computationally cheap and easy to implement on top of current self-tuning methods to improve meta-gradient estimation. Further details are provided in appendix B.

4 Experiments

We demonstrate the catastrophic failure of current meta-gradient approaches in a toy problem, then we scale to a deep RL experiment where we observe that our method improves performance over biased meta-gradients. We additionally provide preliminary results for some environments from the Atari benchmark in Appendix C.3. The code for all experiments is available at `[MASKED-FOR-REVIEW]`.

4.1 Discounting Chain

For our first experiment, we provide a simple illustration of the effects of the bias caused by using the inner critic in the outer loss. For this purpose, we experiment on the Gymnax [15] JAX implementation of the toy Discounting Chain environment originally proposed in bsuite[1] [20]. The Discounting

[1]We use the environment as an illustrative case rather than according to the specification of the bsuite setup.

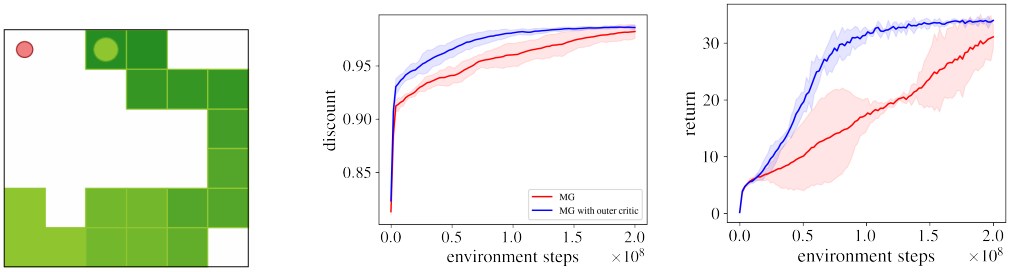


Figure 3: Snake experiment. (a) A visualization of the environment. (b) Discount during training. (c) Return during training. Shaded areas represent standard deviation across 3 seeds.

Chain environment, shown in Figure 2, has a structure designed to highlight the effect of different discount horizons and is therefore highly sensitive to the discount factor. Since we have access to the true value function for this environment, we use it in the policy gradient advantage when training the policy. Having the true value function allows us to focus on the bias induced by using the wrong value of γ in the outer loss. We train the MG and BMG agents with and without the outer critic, with an initial $\gamma = 0.95$. We provide further details on the experimental setup in Appendix C.1. In figure 2, the standard MG and BMG agents fail catastrophically, as they are unable to increase the discount factor and therefore converge to the myopic policy. The reason is that if the inner-critic is used in the outer-loss advantage, then myopic policies are preferred (because $V^{\pi,\gamma} < V^{\pi,\gamma'}$, which then causes the meta-gradient to push γ down instead of up. On the other hand, the MG and BMG agents equipped with the outer critic are able to successfully self-tune the discount factor to converge to the optimal policy and solve the task.

4.2 Snake

Next, we scale our experiments to the deep RL setting using the Snake environment from Jumanji [5, 6]. This environment is sensitive to the discount factor, as the snake agent must plan ahead to maximize return without bumping into itself. We compare an MG algorithm with and without the outer critic. Unlike in the Discounting Chain experiment, where we had access to the true value function, we now train a value function for the inner and outer critic. We give γ a relatively poor initialization of 0.8, as this makes the self-tuning more challenging. We provide further details for the setup of the experiment in Appendix C.2. In figure 3, the agent with the second critic shows a stronger meta-gradient signal, which results in γ being increased faster. Fixing the meta-gradient bias results in a significant improvement in training performance, although both agents asymptotically converge to the maximum return.

5 Conclusion

We have shown that current meta-gradient RL algorithms suffer from a meta-gradient bias induced by having the critic, trained using the meta-learned discount factor, estimate the advantage of the outer objective which necessitates a different discount factor. In our experiments, we have demonstrated that this meta-gradient bias can lead to catastrophic failure of meta-gradient RL algorithms, which we fix with a simple, yet efficient solution. We augment the critic network with an outer critic head whose goal is to estimate the value function discounted by the outer discount factor. Furthermore, we have shown that our method results in performance improvements on more complex environments.

Bootstrapped meta-gradient [8], the state-of-the-art meta-gradient RL agent, has worse than human performance on Bowling, Solaris, and Skiing from the Atari ALE benchmark. These environments are known to be challenging due to delayed reward/credit assignment [3, 2]. Thus, correcting the bias we highlight in this work may be important for improving meta-gradient RL approaches in these increasingly challenging environments, which we hope to study in future work.

References

[1] R. Amit, R. Meir, and K. Ciosek. Discount factor as a regularizer in reinforcement learning, 2020.

[2] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter. Rudder: Return decomposition for delayed rewards. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[3] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell. Agent57: Outperforming the atari human benchmark. *CoRR*, abs/2003.13350, 2020.

[4] 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, jun 2013.

[5] C. Bonnet, D. Byrne, V. Le, L. Midgley, D. Luo, C. Waters, S. Abramowitz, E. Toledo, C. Courtot, M. Morris, D. Furelos-Blanco, N. Grinsztajn, T. D. Barrett, and A. Laterre. Jumanji: Industry-driven hardware-accelerated rl environments, 2022.

[6] C. Bonnet, P. Caron, T. D. Barrett, I. Davies, and A. Laterre. One step at a time: Pros and cons of multi-step meta-gradient reinforcement learning. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021.

[7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

[8] S. Flennerhag, Y. Schroecker, T. Zahavy, H. van Hasselt, D. Silver, and S. Singh. Bootstrapped meta-learning. In *International Conference on Learning Representations*, 2022.

[9] E. Greensmith, P. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[10] A. Gruslys, W. Dabney, M. G. Azar, B. Piot, M. Bellemare, and R. Munos. The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning, 2017.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[12] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, 2015.

[13] L. Kirsch, S. van Steenkiste, and J. Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives, 2019.

[14] V. Konda and J. Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[15] R. T. Lange. gymnax: A JAX-based reinforcement learning environment library, 2022.

[16] B. Liu, X. Feng, H. Zhang, J. Wang, and Y. Yang. A theoretical understanding of gradient bias in meta-reinforcement learning, 2021.

[17] J. Luketina, S. Flennerhag, Y. Schroecker, D. Abel, T. Zahavy, and S. Singh. Meta-gradients in non-stationary environments, 2022.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016.

[19] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih. Monte carlo gradient estimation in machine learning. 2019.

[20] I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvari, S. Singh, B. V. Roy, R. Sutton, D. Silver, and H. V. Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020.

[21] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation, 2015.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[23] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[24] V. Veeriah, M. Hessel, Z. Xu, R. Lewis, J. Rajendran, J. Oh, H. van Hasselt, D. Silver, and S. Singh. Discovery of useful questions as auxiliary tasks, 2019.

[25] R. Vuorio, J. Beck, S. Whiteson, J. Foerster, and G. Farquhar. An investigation of the bias-variance tradeoff in meta-gradients, 2022.

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

[27] Z. Xu, H. van Hasselt, M. Hessel, J. Oh, S. Singh, and D. Silver. Meta-gradient reinforcement learning with an objective discovered online, 2020.

[28] Z. Xu, H. P. van Hasselt, and D. Silver. Meta-gradient reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[29] T. Zahavy, Z. Xu, V. Veeriah, M. Hessel, J. Oh, H. van Hasselt, D. Silver, and S. Singh. A self-tuning actor-critic algorithm, 2020.

[30] Z. Zheng, J. Oh, and S. Singh. On learning intrinsic rewards for policy gradient methods, 2018.

A Background

The goal of an RL agent is to maximize the γ-discounted state value function: $V^{\pi,\gamma}(s) = \mathbb{E}_{\pi|s_t=s}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\right]$.

The action value function is analogously defined below.

$$\begin{aligned} Q^{\pi,\gamma}(s,a) &= \mathbb{E}_{\pi|s_t=s,a_t=a}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\right] \\ &= \mathbb{E}_{\pi|s_t=s,a_t=a}\left[r_t + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\right] \\ &= \mathbb{E}_{\pi|s_t=s,a_t=a}\left[r_t + \gamma V^{\pi,\gamma}(s_{t+1})\right] \end{aligned}$$

The policy gradient theorem [26, 21] in equation 4 provides a way to derive the gradient of the expected discounted return using the value function of the policy.

$$\begin{aligned} \nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_t \gamma^t r_t\right] &= \mathbb{E}_{\pi_\theta}\left[\sum_t \gamma^t r_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\sum_t Q^{\pi,\gamma}(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\sum_t \left(r_t + \gamma V^{\pi,\gamma}(s_{t+1})\right)\nabla_\theta \log \pi_\theta(a_t|s_t)\right] \end{aligned} \tag{4}$$

Actor-critic algorithms [14, 18] learn a parameterized value function (the critic) and a parameterized policy which is updated in a direction informed by the critic. More precisely, the critic learns the value function $V_{\theta_c}^{\pi,\gamma}$ with critic parameters θ_c, and policy π_{θ_p} with policy parameters θ_p. Estimated action value function $\hat{Q}^{\pi,\gamma}(s,a) = r + \gamma V_{\theta_c}^{\pi,\gamma}(s')$ where r and s' are respectively the reward and the next state obtained from taking action a in state s. The goal of an RL agent is to learn the policy π that maximizes the γ-discounted expected return: $\mathbb{E}_{\pi|s_t=s}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k}\right]$, where r_t and s_t are the reward and state at timestep t.

An actor-critic update (A2C) [18] is composed of 3 parts:

- Policy gradient: $\left(r + \gamma V_{\theta_c}^{\pi,\gamma}(s')\right)\nabla_{\theta_p}\log\pi_{\theta_p}(a|s)$
- Critic gradient: $\left(V_{\theta_c}^{\pi,\gamma}(s) - \left(r + \gamma V_{\theta_c}^{\pi,\gamma}(s')\right)\right)\nabla_{\theta_c}V_{\theta_c}^{\pi,\gamma}(s)$
- Entropy regularization: $\nabla_{\theta_p}H(\pi_{\theta_p}(.|s))$

Therefore, the full actor-critic update is: $\theta \leftarrow \theta + \alpha f(\theta,\eta,\tau)$ where $\theta = \{\theta_p,\theta_c\}$ and $\eta = \{\gamma\}$ or $\eta = \{\gamma,\lambda\}$ if TD(λ) is used for advantage estimation, and

$$f(\theta,\eta,\tau) = \left(r + \gamma V_{\theta_c}^{\pi,\gamma}(s')\right)\nabla_{\theta_p}\log\pi_{\theta_p}(a|s) + \left(V_{\theta_c}^{\pi,\gamma}(s) - \left(r + \gamma V_{\theta_c}^{\pi,\gamma}(s')\right)\right)\nabla_{\theta_c}V_{\theta_c}^{\pi,\gamma}(s) + \nabla_{\theta_p}H(\pi_{\theta_p}(.|s))$$

B Outer Critic

To learn the outer value function, we augment the critic network with a second head that outputs the value with discount γ'. To avoid canceling gradients between the two value functions being learned, we use a stop-gradient between the common torso and the outer-critic head (see figure 4. This way, the critic learning dynamic is not changed by adding the outer-critic.

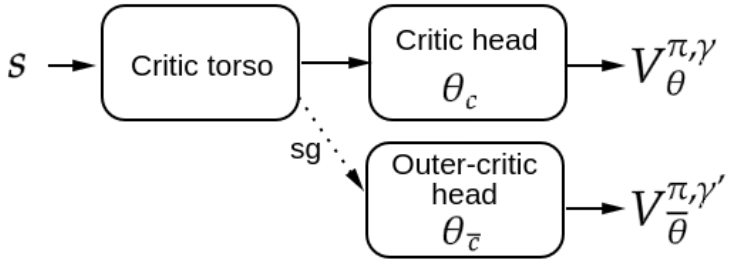


Figure 4: Augmented critic architecture to learn two value functions: the inner and outer values. "sg" means that the `stop_gradient` operation is used to prevent gradients from flowing from the outer-critic head to the common torso.

C Experiments

C.1 Discounting Chain Experiment Details

We initialize $\gamma = 0.95$ such that the value function will initially favor the myopic policy. For simplicity, we use only a policy gradient loss for both the inner and outer loss functions. Furthermore, for BMG we do a single gradient step with the outer loss to obtain the target, which we use for policy matching. We optimize the parameters of the actor for MG and BMG with SGD and use Adam [12] to optimize the discount factor γ.

For all agents, we use a policy network composed of a single linear layer. We generate a new batch of data online for both the inner and outer loss functions. Using Adam to optimize the parameters changes the meta-gradient for BMG, as the momentum causes the target update to be affected by the direction of the inner update. As we would like to our update step to be in the direction of the unbiased meta-gradient, we use SGD instead of Adam.

C.2 Snake Experiment Details

We use the environment "Snake-6x6-v0" from Jumanji [5] by calling Jumanji's registry: `jumanji.make("Snake-6x6-v0")`. In this environment, a *snake* agent navigates a grid in order to eat apples, while avoiding its tail which causes it to die. Hyper-parameters used in the experiments are provided in table 1.

Parameters	**Snake**	**Atari RAM**
architecture	Conv + MLP	MLP
γ^{start}	0.8	0.99
λ	0.95	0.95
learning rate α	3e-4	1e-4
learning rate schedule	constant	constant
inner optimiser	Adam	Adam
gradient clipping norm	None	None
batch size	512	32
num rollouts	5	20
γ'	1	0.995
λ'	0.95	0.995
meta-learning rate β	1e-3	1e-3
meta optimiser	Adam	Adam
meta-gradient clipping norm	None	None

Table 1: Hyper-parameters used in the Snake and Atari experiments.

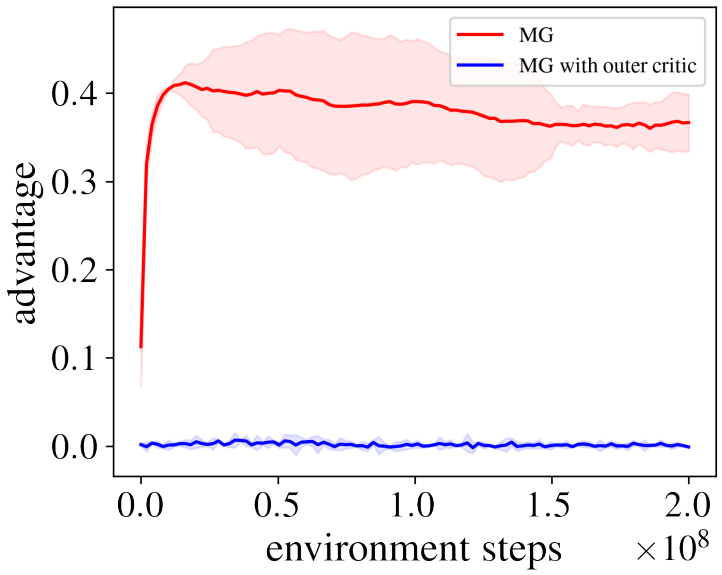


Figure 5: Advantage estimation in the outer loss using the inner critic vs the outer critic. We see that bias caused by using the inner critic in the outer loss results in the advantage being biased such that it is no longer centered on 0 as it should be.

C.3 Preliminary Atari Results

We test our meta-gradient bias correction on some Atari 2600 video games from Arcade Learning Environment (ALE) [4]. These games are known to have long episodes that require careful exploration and credit assignment, which makes self-tuning methods pretty adapted. We use the RAM versions of the environments (learn from RAM memory instead of pixels) to enable faster training.

We train an actor-critic algorithm empowered with meta-gradients (similar setting as [28]) and show training curves in figure 6. The hyper-parameters used in the experiments are gathered in table **??**.

We can see in figure 6 that the outer advantage bias catastrophically pushes the discount factor down, which hinders agent performance. Our method fixed the bias, achieves higher performance, and prevents myopic self-tuning.

It is to be noted that self-tuning in these environments is very noisy and hard to achieve on a consistent basis. Although fixing this meta-gradient bias seems to help during training, making self-runing more consistent and stable is an interesting research direction.

Figure 6: Discounting chain experiment. **(Top)**: *Berzerk* game. **(Middle)**: *JamesBond* game. **(Bottom)**: *Bowling* game. **(Left)**: average episode return. **(Middle)**: discount factor during training. **(Right)**: outer-loss advantage. Having a non-zero advantage average highlights the bias in the advantage estimation.