On the reproducibility of gradient-based Meta-Reinforcement Learning baselines

Tristan Deleu, Simon Guiroy, Seyedarian Hosseini
MILA – Université de Montréal
Montreal, QC, Canada
`tristan.deleu@gmail.com`

Abstract

Meta-learning provides an appealing solution to the data-efficiency issue inherent in both deep supervised learning and (model-free) deep reinforcement learning. The diversity of tasks available in supervised meta-learning and meta-reinforcement learning enabled the fast progress we are recently observing in this field, since one can easily compare a new meta-learning method to existing algorithms. In this paper, we revisit one of these baselines on two basic meta-reinforcement learning problems: the multi-armed bandits and tabular MDPs. We provide updated results for MAML applied to these two problems, and show that MAML compares favorably to more recent meta-learning approaches, contrary to what was previously reported. Along with this baseline, we also include some new results on the same tasks for Reptile, a first-order meta-learning approach.

1 Introduction

Just like in any other subfield of machine learning, having access to standard benchmarks is crucial to fairly compare the performance of different meta-learning algorithms. Supervised meta-learning benefits from well-established benchmarks in the few-shot learning literature, such as classification problems on Omniglot [7] or mini-Imagenet [17, 12]. Likewise, some challenging tasks like continuous control problems [3, 16] and navigation from visual inputs [2, 9] have also been proposed in the meta-reinforcement learning (meta-RL) literature. On the other end of the spectrum, the authors of [2] also introduced some basic meta-RL tasks, based on classical RL problems such as multi-armed bandits and tabular MDPs.

The authors of [9] used this same suite of classical RL problems to assess the performance of their meta-learning algorithm (SNAIL), and also reported some (partial) baseline results based on another meta-learning algorithm called MAML [3]. In this paper, we provide updated results for the MAML baseline on the full suite of classical RL problems, and we show that MAML compares favorably to SNAIL, contrary to what was reported in [9]. In addition to reproducing this baseline, we also include new results on the same suite for a recently proposed first-order meta-learning algorithm, called Reptile [11].

2 Background

Reinforcement Learning In supervised meta-learning, a task $\mathcal{T}$ could be, for example, a classification problem. In the context of meta-RL, a task $\mathcal{T} = \langle \mathcal{S}, \mathcal{A}, p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}), r(\mathbf{s}, \mathbf{a}, \mathbf{s}') \rangle$ is typically a Markov Decision Process (MDP), where $\mathcal{S}$ is the set of states, and $\mathcal{A}$ the set of actions. $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ is the probability of transition from a state $\mathbf{s} \in \mathcal{S}$ to a state $\mathbf{s}'$ after taking action $\mathbf{a} \in \mathcal{A}$. Following this transition, the agent receives a reward $r(\mathbf{s}, \mathbf{a}, \mathbf{s}')$. For some discount factor $\gamma \in [0, 1]$, the return G_t

Preprint. Work in progress.

at time t is a random variable corresponding to the discounted sum of rewards observed after t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

where R_{t+1} is the (random) reward received after taking action A_t in state S_t. A policy π is a mapping from the states to the probabilities of each actions in $\mathcal{A}$, ie. $\pi(\mathbf{a} \mid \mathbf{s})$ is the probability of selecting the action $\mathbf{a}$ while being in state $\mathbf{s}$. This policy π_θ might be parametrized by some θ, and the goal of policy-gradient methods is to find a set of parameters θ that maximize the expected return, or state-value function $v_{\pi_\theta}(\mathbf{s})$

$$v_{\pi_\theta}(\mathbf{s}) = \mathbb{E}_{\pi_\theta}[G_t \mid S_t = \mathbf{s}] \tag{2}$$

While the framework of meta-RL allows for different tasks to have different sets of states and actions, we will restrict ourselves to fixed sets of states $\mathcal{S}$ and actions $\mathcal{A}$ across tasks in this paper. Only the transition probabilities $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ and rewards $r(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ may vary from one task to another.

Meta-Learning Following the coarse categorization proposed in [4], one way to approach meta-learning is through *parameters adaptation*. Given a model $f(\mathbf{x}; \theta)$ parametrized by a vector θ, we want to find some parameter $\theta'_{\mathcal{T}}$ adapted to the task $\mathcal{T}$, so that the learner f generalizes well on task $\mathcal{T}$. That is, given some data $\mathcal{D}_{\mathcal{T}}$ from task $\mathcal{T}$[1], the adapted parameter $\theta'_{\mathcal{T}}$ is returned by the meta-learner $g(\mathcal{D}_{\mathcal{T}}; \phi)$, which might have some parameters ϕ. Note that in meta-RL, the learner f corresponds to a (parametrized) policy. Then for some new test input $\mathbf{x}^\star$, the prediction $\hat{\mathbf{y}}^\star$ is defined as

$$\hat{\mathbf{y}}^\star = f(\mathbf{x}^\star; \theta'_{\mathcal{T}}) = f(\mathbf{x}^\star; g(\mathcal{D}_{\mathcal{T}}; \phi)) \tag{3}$$

This formulation embraces a wide variety of meta-learning algorithms (see [8] for an overview of some of these methods), where the meta-learner g could be, for example, a neural network itself [1, 12]. In particular, this also includes *gradient-based methods*, where the meta-learner g is a fixed gradient descent learning rule. In this paper, we focus on two of these gradient-based algorithms, MAML [3] and Reptile [11], which we briefly recall in the following section.

Finally, the objective we want to optimize is the expected loss of this adapted parameter $\theta'_{\mathcal{T}}$ over a distribution of tasks $p(\mathcal{T})$

$$\min_{\phi} \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})}[\mathcal{L}_{\mathcal{T}}(\theta'_{\mathcal{T}}; f)] = \min_{\phi} \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})}[\mathcal{L}_{\mathcal{T}}(g(\mathcal{D}_{\mathcal{T}}; \phi); f)] \tag{4}$$

where the loss $\mathcal{L}_{\mathcal{T}}$ could be, for example, derived from the state-value function in Equation (2).

3 Gradient-based meta-learning

3.1 Model-Agnostic Meta-Learning

Taking inspiration from fine-tuning, Model-Agnostic Meta-Learning (MAML, [3]) is a meta-learning algorithm that learns an initial set of parameters θ of the policy π, so that only a few gradient steps, starting from θ, are required to adapt the parameters $\theta'_{\mathcal{T}}$ to a new task $\mathcal{T}$. In practice, we can limit ourselves to a single gradient step with step size α.

$$g(\mathcal{D}_{\mathcal{T}}; \theta) = \theta - \alpha \nabla_\theta \mathcal{L}(\theta; \mathcal{D}_{\mathcal{T}}) \tag{5}$$

This method is *model-agnostic*, meaning that the policy π can be parametrized by any neural network. In the context of reinforcement learning, the gradient update can be computed using vanilla policy-gradient (REINFORCE, [18]). If N trajectories $\mathcal{D}_{\mathcal{T}} = \{(\mathbf{s}_0^{(i)}, \mathbf{a}_0^{(i)}, \mathbf{s}_1^{(i)}, \mathbf{a}_1^{(i)}, \ldots)\}_{i=1}^N$ have been sampled through interactions with the task $\mathcal{T}$, then the inner-loss $\mathcal{L}$ can be defined as

$$\mathcal{L}(\theta; \mathcal{D}_{\mathcal{T}}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{\infty} \gamma^t G_t^{(i)} \log \pi(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)}; \theta) \tag{6}$$

The pseudo-code for MAML is given in Algorithm 1 (adapted from Algorithm 3 in [3]). We use Trust Region Policy Optimization (TRPO, [13]) as the meta-optimizer for Equation (4) (line 10 in Algorithm 1).

[1]Meta-learning typically operates in the *low-data regime*, and $\mathcal{D}_{\mathcal{T}} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ could be a (small) training set for task $\mathcal{T}$ in the case of few-shot supervised learning. Similarly in meta-RL, $\mathcal{D}_{\mathcal{T}}$ is usually a small set of trajectories obtained through interactions with the task (ie. MDP) $\mathcal{T}$.

Algorithm 1 MAML for Reinforcement Learning

```
Require: $p(\mathcal{T})$ distribution over tasks $\mathcal{T}$.
Require: $\alpha$, $\beta$ step size hyperparameters.
 1: Randomly initialize $\theta$
 2: while not done do
 3:     Sample batch of $n$ tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:     for all $\mathcal{T}_i$ do
 5:         Sample $\mathcal{D}_i = \{(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_H)\}$ $N$ trajectories from task $\mathcal{T}_i$, following policy $\pi_\theta$
 6:         Evaluate $\mathcal{L}_{\mathcal{T}_i}(\pi_\theta)$ with $\mathcal{D}_i$, Equation (6)
 7:         Compute the adapted parameters with 1 step of gradient descent: $\theta'_i = \theta - \alpha\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\pi_\theta)$
 8:         Sample $\mathcal{D}'_i = \{(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_H)\}$ trajectories from task $\mathcal{T}_i$, following policy $\pi_{\theta'_i}$
 9:     end for
10:     Update $\theta \leftarrow \theta - \frac{\beta}{n}\nabla_\theta \sum_{i=1}^{n} \mathcal{L}_{\mathcal{T}_i}(\pi_{\theta'_i})$ using each $\mathcal{L}_{\mathcal{T}_i}$ evaluated on $\mathcal{D}'_i$
11: end while
```

Algorithm 2 Reptile for Reinforcement Learning

```
Require: $p(\mathcal{T})$ distribution over tasks $\mathcal{T}$.
Require: $\alpha$, $\beta$ step size hyperparameters.
 1: Randomly initialize $\theta$
 2: while not done do
 3:     Sample batch of $n$ tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:     for all $\mathcal{T}_i$ do
 5:         Sample $\mathcal{D}_i = \{(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_H)\}$ $N$ trajectories from task $\mathcal{T}_i$, following policy $\pi_\theta$
 6:         Evaluate $\mathcal{L}_{\mathcal{T}_i}(\pi_\theta)$ with $\mathcal{D}_i$, Equation (6)
 7:         Compute the adapted parameters $\theta'_i$ with $k$ steps of gradient descent on $\mathcal{L}_{\mathcal{T}_i}$ with step $\alpha$
 8:
 9:     end for
10:     Update $\theta \leftarrow \theta - \frac{\beta}{n} \sum_{i=1}^{n} (\theta - \theta'_i)$
11: end while
```

3.2 Reptile: First-order meta-learning

One of the main drawbacks of MAML is that it requires the computation of second-order derivatives of the inner-loss function $\mathcal{L}$ during the meta-optimization. In order to avoid this expensive computation, a first-order alternative to MAML has been proposed, called Reptile [11]. Similar to MAML, the adaptation of the parameters to a new task $\mathcal{T}$ is performed through k gradient steps, with step size α (where typically $k > 1$ here, contrary to MAML). The meta-learner g is therefore defined as

$$g(\mathcal{D}_\mathcal{T}; \theta) = \theta_\mathcal{T}^{(k)} \qquad \text{where} \qquad \begin{aligned} \theta_\mathcal{T}^{(0)} &= \theta \\ \theta_\mathcal{T}^{(t+1)} &= \theta_\mathcal{T}^{(t)} - \alpha \nabla_{\theta_\mathcal{T}^{(t)}} \mathcal{L}(\theta_\mathcal{T}^{(t)}; \mathcal{D}_\mathcal{T}) \end{aligned}$$

The key difference between Reptile and MAML is in the way the parameters θ get updated for the meta-optimization in Equation (4). The meta-update is given by

$$\theta \leftarrow \theta - \frac{\beta}{n} \sum_{i=1}^{n} (\theta - g(\mathcal{D}_\mathcal{T}; \theta)) \tag{7}$$

where β is another step size hyperparameter for the meta-optimizer. In practice, we can use more advanced first-order optimizers such as Adam [6] instead of a standard gradient descent update. The pseudo-code for Reptile is given in Algorithm 2 (side-by-side with Algorithm 1 to emphasize the similarities between these two meta-learning algorithms).

4 Experiments

4.1 Multi-armed bandits

In our K-armed bandits experiments (adapted from [2]), each of the K arms gives a reward sampled from a Bernoulli distribution with parameter $p \in [0, 1]$. The goal here is to train an agent that is able to adapt to a new task, given N realizations of this task (ie. N rewards in $\{0, 1\}$).

In addition to the meta-learning results, we also include the Gittins index, as reported in [2]. The Gittins index is the Bayes optimal solution in the discounted, infinite horizon setting [5]. Note that since this index is only optimal as $N \to \infty$, a meta-RL agent could choose to exploit sooner, and outperform it for smaller values of N. We report the mean total rewards obtained on both MAML and Reptile for different settings of K (number of arms) and N (number of observations) in Table 1, along with their 95% confidence intervals.

The first observation we can make from Table 1 is that we obtained significantly higher returns in our reproduction of the MAML baseline than the ones reported in [9] (when available). They found that training MAML was too expensive for $N = 500, 1000$. In order to scale to larger problems

Setup		Gittins	As reported in [9]			Ours	
N	K	(optimal as $N \to \infty$)	Random	MAML	SNAIL	MAML	Reptile
10	5	6.6	5.0	6.5 ± 0.1	6.6 ± 0.1	7.0 ± 0.1	$\mathbf{7.2 \pm 0.2}$
10	10	6.6	5.0	6.6 ± 0.1	6.7 ± 0.1	6.8 ± 0.1	$\mathbf{7.1 \pm 0.3}$
10	50	6.5	5.1	6.6 ± 0.1	$\mathbf{6.7 \pm 0.1}$	$\mathbf{6.7 \pm 0.3}$	6.6 ± 0.3
100	5	78.5	49.9	67.1 ± 1.1	79.1 ± 1.0	80.3 ± 0.4	$\mathbf{81.6 \pm 1.9}$
100	10	82.8	49.9	70.1 ± 0.6	83.5 ± 0.8	$\mathbf{85.2 \pm 0.5}$	84.8 ± 0.6
100	50	85.2	49.8	70.3 ± 0.4	$\mathbf{85.1 \pm 0.6}$	83.5 ± 0.8	82.9 ± 1.2
500	5	405.8	249.8	–	408.1 ± 4.9	$\mathbf{416.8 \pm 0.6}$	412.0 ± 7.3
500	10	437.8	249.0	–	432.4 ± 3.5	$\mathbf{448.0 \pm 1.0}$	444.5 ± 7.4
500	50	463.7	249.6	–	442.6 ± 2.5	446.8 ± 2.2	$\mathbf{454.0 \pm 6.1}$
1000	50	944.1	499.8	–	889.8 ± 5.6	909.2 ± 7.2	$\mathbf{920.8 \pm 7.2}$

Table 1: Average cumulative reward for multi-armed bandit tasks. The evaluation is detailed in Appendix B. Along with our results, we also include the ones reported in [2, 9].

with $N \geq 500$, we decoupled the number of parallel workers that interact with task $\mathcal{T}$ from the number of trajectories sampled N. This means that a single worker could sample (serially) multiple trajectories from task $\mathcal{T}$. Moreover, these results on MAML appear to be surprisingly better than SNAIL introduced in [9], even though the policy we used in MAML has a lower capacity than SNAIL (see Appendix A for details). We also often get close to the Gittins index, even for larger values of N. In our experiments, Reptile seems to overall perform as well as MAML on multi-armed bandits.

4.2 Tabular MDPs

In our Tabular MDPs experiments (adapted from [2]), each task corresponds to a specific transition matrix and reward function. All the MDPs in our experiments have $|\mathcal{S}| = 10$ states, $|\mathcal{A}| = 5$ actions, and their reward functions are Gaussian distributed with unit variance. The rows of the transition matrix are sampled according to a flat Dirichlet(1) distribution. The mean values of the reward function for task $\mathcal{T}$ are sampled from a Gaussian distribution $\mathcal{N}(1, 1)$. The goal is to train an agent that is able to adapt to a new task, given N trajectories of length $H = 10$. We report the mean cumulative rewards obtained on both MAML and Reptile for different settings of N (number of episodes) in Table 2, along with their 95% confidence intervals.

Setup	As reported in [9]			Ours	
N	Random	MAML	SNAIL	MAML	Reptile
10	0.482	0.563	$\mathbf{0.766 \pm 0.001}$	0.745 ± 0.027	0.706 ± 0.012
25	0.482	0.591	$\mathbf{0.862 \pm 0.001}$	$\mathbf{0.859 \pm 0.012}$	0.813 ± 0.009
50	0.481	–	0.908 ± 0.003	$\mathbf{0.912 \pm 0.020}$	0.844 ± 0.090
75	0.482	–	0.930 ± 0.002	$\mathbf{0.936 \pm 0.024}$	0.850 ± 0.078
100	0.481	–	0.941 ± 0.003	$\mathbf{0.944 \pm 0.019}$	0.826 ± 0.102

Table 2: Average cumulative reward for the tabular MDP tasks, normalized by the average reward achieved by value iteration. The evaluation is detailed in Appendix B. Along with our results, we also include the ones reported in [9].

Similar to the multi-armed bandits experiments, we found that our reproduction of the MAML baseline produces significantly better results compared to the ones reported in [9]. We were also able to scale our experiments to larger values of N. Our results on MAML also match the performance of SNAIL on tabular MDPs, like it did on multi-armed bandits. However this time, Reptile does not perform as well as MAML. We conjecture that this gap in performance might be explained by a high variance during meta-training, which translates in bigger confidence intervals in evaluation, especially for larger values of N.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474, 2016.

[2] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL2: Fast Reinforcement Learning via Slow Reinforcement Learning. *CoRR*, abs/1611.02779, 2016.

[3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *CoRR*, abs/1703.03400, 2017.

[4] Chelsea Finn and Sergey Levine. Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm. *CoRR*, abs/1710.11622, 2017.

[5] John C Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 148–177, 1979.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[7] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[8] Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Learning Unsupervised Learning Rules. *CoRR*, abs/1804.00222, 2018.

[9] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. Meta-Learning with Temporal Convolutions. *CoRR*, abs/1707.03141, 2017.

[10] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *CoRR*, abs/1602.01783, 2016.

[11] A. Nichol, J. Achiam, and J. Schulman. On First-Order Meta-Learning Algorithms. *ArXiv e-prints*.

[12] Sachin Ravi and Hugo Larochelle. Optimization as a Model for Few-Shot Learning. 2016.

[13] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *CoRR*, abs/1502.05477, 2015.

[14] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *CoRR*, abs/1506.02438, 2015.

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. 2018.

[16] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[17] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching Networks for One Shot Learning. *ArXiv e-prints*, June 2016.

[18] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256, 1992.

A Experimental settings

In all our experiments, the learner (policy) π is a 2-layer MLP, each layer having 32 hidden units and followed by a ReLU activation function. While the original paper [3] suggests that they use REINFORCE to compute the inner-loss $\mathcal{L}$ (Equation (6)), in practice the inner-loss used in the source code released alongside the paper is the Advantage Actor-Critic [10, 15]. We also evaluated the inner-loss using A2C, with a linear critic similar to the one used in [2], and we used Generalized Advantage Estimation (GAE, [14]) to estimate the advantage. We used a step size $\alpha = 0.5$ for MAML and $\alpha = 0.1$ for Reptile in both experiments.

We used Trust Region Policy Optimization (TRPO, [13]) as the meta-optimizer (outer-loss) for MAML, on mini-batches of $n = 20$ tasks. We used Adam [6] with learning rate $\beta = 10^{-4}$ as the meta-optimizer for Reptile, again on mini-batches of $n = 20$ tasks. The number of gradient updates for the parameters adaptation in Reptile is $k = 3$. In all our experiments, the discount factor $\gamma = 0.95$.

B Evaluation

In order to evaluate our models, we need to have access to a set of *meta-test* tasks that the agent never interacted with during meta-training. In both experiments, we sampled tasks during meta-training from an infinite pool of tasks, since the distribution over tasks $p(\mathcal{T})$ is equivalent to continuous distributions (Uniform$([0, 1]^K)$ for multi-armed bandits, Dirichlet$(1) \otimes \mathcal{N}(\mathbf{1}_K, \mathbf{I}_K)$ for tabular MDPs). Therefore, we similarly sampled n' new tasks from this this same pool of tasks, since we are guaranteed to sample different meta-test tasks almost surely.

In all our experiments, we evaluated our methods on $n' = 200$ meta-test tasks and 5 different meta-learning agents (ie. 5 different random seeds).