# First-order Meta-Learned Initialization for Faster Adaptation in Deep Reinforcement Learning

**Abhijat Biswas**
Carnegie Mellon University
Pittsburgh, PA 15213
`abhijat@cmu.edu`

**Shubham Agrawal**
Carnegie Mellon University
Pittsburgh, PA 15213
`shubhamag@cmu.edu`

## Abstract

Deep neural networks excel in regimes with large amounts of data, but tend to struggle when data is scarce or when they need to adapt quickly to changes in the task. Recent works have proposed training a meta-learner on a distribution of similar tasks, in the hope of generalization to novel but related tasks by learning a high-level strategy that captures the essence of the problem it is asked to solve. The hypothesis is that is should be possible to learn something about the optimization process while learning on a specific task that generalizes across tasks.

However, recent meta-learning approaches are hand-designed, either using architectures specialized to a particular application, or hard-coding algorithmic components that constrain how the meta-learner solves the task. While there do exist model-agnostic meta-learning methods, they often contain gradient-of-gradient (second order derivative) terms in the parameter update rules. We aim to characterize the effect of first-order approximations to recently proposed meta-learning algorithms that work around this by using the gradient updates of the learner itself. We also evaluate these first-order methods (which have hitherto only been tested in supervised settings) for reinforcement learning problems.

## 1 Introduction

A key difference between human intelligence and the current state of AI is the capability of humans to quickly learn new skills, or recognize objects after just seeing a few examples [1]. Current AI agents, trained with traditional supervised-learning or reinforcement learning methods, are not flexible enough to adapt to a small amount of new data quickly, while avoiding overfitting on it. Humans are able to do so effectively by drawing upon their vast amounts of prior knowledge and experiences. Rather than learning from scratch, they are using world knowledge to adapt to the requirements of a new task quickly.

This makes real world deployment of autonomous agents in their current state difficult because of dynamically changing environments offering a diverse range of task affordances and requiring a wide range of interactions that is not possible to program. In Deep Reinforcement Learning literature, there have been broadly two approaches to solving this problem. One, to transfer knowledge from trained "expert" networks or use shared weights to train a network quickly on a wide variety of tasks. The alternative approach is the Meta-learning framework, which aims at finding ways to optimize the optimization process itself, so as to make the final model quick to adapt given only a few examples from new tasks.

### 1.1 Shared-Learning approaches

In the Transfer learning setting, we aim to utilize the "knowledge" in networks that have proven to do well on some task and try to adapt it for a separate but potentially related task. Transfer learning as an approach has proved to be extremely effective in the supervised learning setting, where, for example, a CNN trained for classification on Imagenet can be fine-tuned for a wide variety of vision-related tasks with massive boosts to training speed [12]. In the Deep Reinforcement Learning setting it has been shown that many game-specific expert networks trained on Atari games can be used to guide the learning of a single multi-task policy network [3].

### 1.2 Meta-Learning approaches

The idea of "learning-to-learn", or meta-learning is not a new one [5] [6]. Meta-learning seeks to broaden the scope of the learning framework to a distribution of related tasks. Rather than training the learner on a single task (with the goal of generalizing to unseen samples from a similar data distribution) a meta-learner is trained on a distribution of similar tasks. The goal in the meta-learning framework is of learning a strategy that generalizes to related but unseen tasks from a similar task distribution.

A common strategy for meta-learning is to train a meta-learner, that in turn learns how to update the parameters of the learner's model. Recently this strategy has been applied to deep nets as well [9] [10]. This strategy was extended in [8] , using a LSTM meta-learner and CNN based classifier as the "learner". In their approach, the meta-learning algorithm had 2 parts: The learner's initialization is learnt for quick adaptation, and the meta-learner is trained to optimize the leaner in meta-learning tasks.

More recently Finn et. al. proposed algorithm, Model-agnostic Meta-Learning (MAML) [4], which uses the gradients of the learner itself for meta-learning. This makes the algorithm model and task-agnostic. Thus any model (trained with gradient-descent) can be combined with the algorithm, for tasks ranging from classification to reinforcement learning.

Another recently proposed approach for meta-learning and few-shot learning in supervised settings is Reptile [2]. The algorithm learns a parameter initialization by repeatedly sampling a task, training on it and moving the initialization toward the trained weights on that task. The authors show that for few-shot supervised classification on the Omniglot [7] and Mini-Imagenet dataset [8], the algorithm can be just as effective as MAML. However, no results are presented in a reinforcement setting.

As evident, these methods often use gradient of gradient methods for meta-optimization that involve second derivatives (second-order), which is often computationally expensive [4]. **In this work, we investigate the performance of first-order approximations to second-order meta-learning methods on multiple tasks in a reinforcement learning setting. We use these meta-learned model parameters as initializations, which can be used to adapt to new reinforcement learning tasks with very few examples.**

Both MAML and Reptile are interesting because they aim to solve the meta-learning problem, not by learning a separate meta-learner, but by gradient based updates to the same learner.

**We aim to investigate the feasibility of first order specifically for meta-learning in the reinforcement learning setting**, where the efficacy of these methods is still unexplored. While MAML showed that second order optimization methods are able to optimize on the meta-problem on RL problems, and Reptile showed that first-order methods are reasonably close to MAML on few-shot learning tasks in a supervised setting, neither has shown task adaptation in the reinforcement learning setting with a first order method.

## 2 Experiments

The key goal was to study whether whether first order methods like Reptile and FOMAML would be as effective as regular MAML or other recent meta-learning approaches in the deep reinforcement learning setting. This involves:

1. Implementing the FOMAML and Reptile algorithms in a Deep reinforcement learning setting with a policy gradient based network

2. Training these algorithms on tasks sampled from environments mentioned in 2.1 and then evaluating them for previously unseen tasks
3. Obtaining quantitative data, to compare the adaptability of these algorithms to baseline methods such as conventional pre-training, regular MAML and random initialization.
4. Quantifying the computational speedup obtained by using the first-order approximation.

## 2.1 Testing Environments

The training phase of the the MAML and Reptile algorithms involves training them in parallel over a sample of tasks drawn from a distribution. At test time, the network is presented with a previously unseen task, and should ideally be able to reach the max performance with a couple of gradient updates. For ease of comparison to baselines, we propose to use the same 3 RL tasks as [4]. Namely, we wish to use:

1. 2D point-mass
   A task in this environment constitutes traveling from a start position to an unknown goal position in a unit square, both of which are randomly sampled. The observation is the current 2D position. The actions are velocity commands in the range $[-0.1, 0.1]$. The reward is the negative of the absolute distance to goal.
2. MuJoCo "Ant" which requires controlling a 4-legged creature in a specific direction or at a specific velocity

The latter has two locomotion-based tasks, making it more complex than than 1.

1. Goal direction: The agent is required to go in either the forward or backward direction (uniformly randomly picked). The reward is the magnitude of velocity in the picked direction.
2. Goal velocity: The agent is required to achieve an unknown (to the agent) goal velocity $\sim U[0, 2]$. The reward is the negative absolute difference between the current velocity of the agent and the goal velocity. A positive reward bonus is added at each timestep to prevent the ant from ending the episode.

For each of these, the policy is a Gaussian MLP with a mean parameterized by a neural network with two hidden layers with 100 neurons each.

## 3 Methods

The generic algorithm for the MAML algorithm proposed in [4] is given in Algorithm 1 . The inner loop gradient updates were computed using vanilla policy gradient (REINFORCE) [15]. The outer, meta-optimizer was Trust-region Policy optimization [11]. A significant computational expense in MAML, comes from the calculation of second derivatives while backpropagating the meta-gradient, through the inner loop gradient operator in the meta-objective. We further characterize this in Figure 3, which shows that we obtain a $50\%$ speedup on average, in the meta-optimization step from the first order approximation.

In the same paper, the authors also proposed a first order approximation of the generic MAML algorithm, called FOMAML, which omitted the second derivatives. While it doesn't contain second-derivative terms, the resulting method still computes the meta-gradient at the post-update parameter values $\theta_i$. Interestingly, the performance of this method is nearly the same as that obtained with full second derivatives, suggesting that majority constituent of the MAML improvement is from the gradients of the objective at the post-update parameter values and that the second order updates from differentiating through the gradient updates are less significant. However, they only demonstrate the use of the FOMAML algorithm on supervised learning tasks.

The efficacy of the first order approximation can be hypothesized on the basis of past works [13], which show that for ReLU non-linearities, locally neural networks behave like linear systems, because of the piece-wise linear nature of ReLU. This suggests that second derivatives through such networks will have very small magnitudes.

The Reptile algorithm in [2] drew on this insight from FOMAML to propose a simplification of the update rule. We include the proposed batch version of the Reptile algorithm is given in Algorithm 3 for comparison's sake.

**Algorithm 1** Generic Model-agnostic Meta-learning algorithm

```
Require: $p(\tau)$ : Distribution over tasks
Require: $\alpha, \beta$ : step hyper-parameters
   Initialisation : Random $\theta$
1: for $i = 1, 2...n$ do
2:     sample tasks $\tau_i \sim p(\tau)$
3:     for all $\tau_i$ do
4:         Evaluate $\nabla_\theta L(\theta)$ with K examples
5:         Compute adapted param with GD: $\theta_i = \theta - \alpha\nabla_\theta L(\theta)$
6:     end for
7:     update: $\theta \leftarrow \theta - \beta\nabla_\theta L(\theta - \alpha\nabla_\theta L(\theta))$
8: end for
```

**Algorithm 2** Generic First-order Model-agnostic Meta-learning algorithm

```
Require: $p(\tau)$ : Distribution over tasks
Require: $\alpha, \beta$ : step hyper-parameters
   Initialisation : Random $\theta$
1: for $i = 1, 2...n$ do
2:     sample tasks $\tau_i \sim p(\tau)$
3:     for all $\tau_i$ do
4:         Evaluate $\nabla_\theta L(\theta)$ with K examples
5:         Compute adapted param with GD: $\theta_i = \theta - \alpha\nabla_\theta L(\theta)$
6:     end for
7:     update: $\theta \leftarrow \theta - \beta\nabla_{\theta_i} L(\theta_i)$
8: end for
```

**Algorithm 3** Batched version of Reptile algorithm

```
Require: $p(\tau)$ : Distribution over tasks
Require: $\alpha, K$ : step hyper-parameters
   Initialisation : Random $\theta$
1: for $i = 1, 2...n$ do
2:     sample tasks $\tau_i \sim p(\tau)$
3:     for all $\tau_i$ do
4:         Evaluate the update $\theta_i = \theta - \alpha\nabla_\theta L_{\tau_i}(\theta)$ $k$ times
5:     end for
6:     update: $\theta = \theta + \frac{\alpha}{K}\sum_i^n(\theta_i - \theta)$
7: end for
```

## 3.1 Analysis of meta-objective and the first order approximation

In particular, we are interested in the comparison of the meta-objectives in MAML and FOMAML.

Notation:

$$U_\tau^k \rightarrow k \text{ SGD updates on example(s) sampled from task} \tau$$
$$L_\tau \rightarrow \text{Loss function on task } \tau$$

The MAML update rule then is:

$$\text{minimize}_\theta \; \mathbb{E}_\tau \left[ L_\tau(U_\tau^1(\theta)) \right]$$

To compute this update, we compute the gradient w.r.t the parameters $\theta$.

$$\begin{aligned} g_{\text{MAML}} &= \frac{\partial}{\partial\theta} \left[ L_\tau(U_\tau^1(\theta)) \right] \\ &= U_\tau^{1'}(\theta) L_\tau^{'}(\theta_{\text{new}}) \end{aligned}$$

Here, $\theta_{\text{new}} = U_\tau^k(\theta)$. $U^{k'}$ is the Jacobian of the update operation w.r.t the parameters $\theta$.

In first-order MAML (FOMAML), the proposed update computes the derivative at the post-update parameter values, without differentiating through the derivative operator in the inner loop update. For instance, in the reinforcement learning version of FOMAML (see: 2):

1. sample N tasks from the space of tasks
2. for each task $\tau_i$, compute one iteration of policy gradient, leading to new policy parameters $\theta_i$
3. compute the meta-gradient at $\theta_i$ (rather than $\theta$ as in MAML)
4. average the N such gradients obtained from the N tasks

In context of the above mathematical formulation, this approximation is equivalent to setting the Jacobian of the SGD update to the identity matrix. For a particular task, $\tau$ this yields:

$$\begin{aligned} g_{\text{FOMAML}} &= \frac{\partial}{\partial\theta_i} \left[ L_\tau(U_\tau(\theta)) \right] \\ &= L_\tau^{'}(\theta_i) \end{aligned}$$

Then, the gradient can be plugged directly into the update equation:

$$\theta \leftarrow \theta - \beta \cdot g_{\text{FOMAML}}$$

For a batch of tasks, the gradient is simply summed over all tasks.

Essentially, the difference between MAML and FOMAML is that FOMAML computes the meta-objective derivative at the post policy gradient parameters, and hence doesn't have to compute the derivative through the policy gradient . For a more detailed mathematical treatment of this comparison, see [2].

At test time, these models are treated as initialization for the

## 4 Results

We present the results of the oracle (agent that has access to the goal as well as the current state) together with the MAML policy (2nd-order) in the 2D point-mass environment in Fig 6 and the MuJoCo Ant tasks in Fig 4 and Fig5. It can be seen that both MAML and FOMAML are able to achieve oracle-like performance within a couple of gradient update steps on unseen tasks.

For both environments, we use a meta-learning rate of $0.01$ and an inner loop learning rate of $0.1$. At test time, updates were performed using vanilla policy gradient with linear baseline and a learning rate of $0.1$.

## 4.1 Pointmass environment

We first compare the performance metrics of FOMAML and MAML on the 2D pointmass environment. We performed most of our experiments on this task. To determine the effect of the various hyper-parameters, we performed ablation studies on the 2D point-mass environment. The two most important hyper-parameters were the meta-batchsize and the number of tasks sampled per meta-batch (we call this the fast-batch size).

For both MAML and FOMAML, performance improves as either the batch size or number of tasks increases. While with different settings, they both outperform each other almost an equal number of times, most of the differences are not statistically significant. This is encouraging, as it suggests we can use first-order approximations without sacrificing accuracy in adaptability to new tasks.

This is also in accordance with the results from the supervised meta learning experiments in [2] and [4].

Figures 1 and 2 show the performance variation.

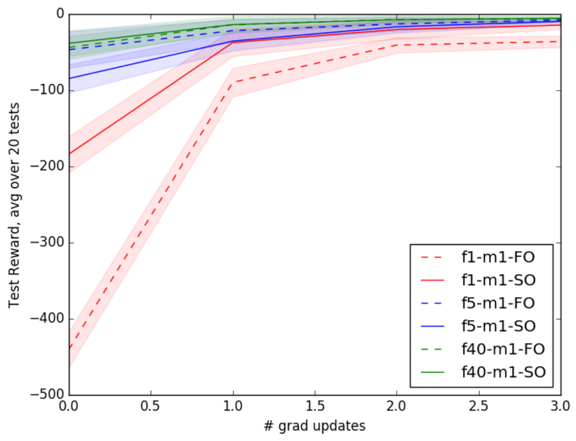

Figure 1: Ablation on number of tasks sampled in the inner loop in the Pointmass environment. f-fast batch size; m-meta batch size; FO/SO-first order/second order

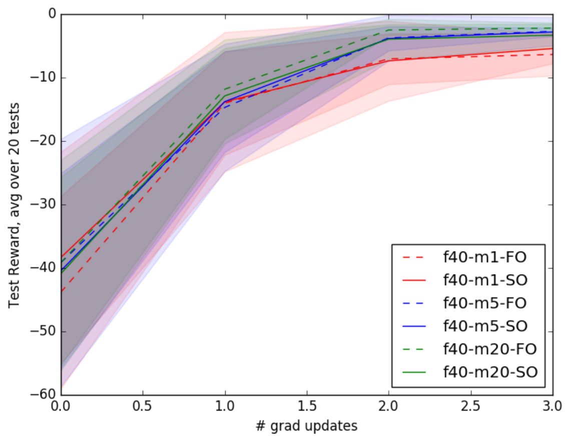

Figure 2: Ablation on meta-batch size in the Pointmass environment. f-fast batch size; m-meta batch size; FO/SO-first order/second order

## 4.2 MuJoCo Ant environment

Our next set of meta-RL experiments are in the MuJoCo simulator [16], on the Ant locomotion tasks. We evaluate on two types of tasks, hereon referred to as "direction" and "velocity". In the direction tasks, a direction is picked at random and the reward is based on magnitude of velocity in that direction. In the velocity task, the agent is required to attain a goal velocity and the rewards are based on the difference between the agent's current velocity and the target. A positive reward bonus is added at each time-step to prevent the ant from ending the episode.
The performance plots are visualized in fig 5 and fig 4 for the direction and velocity tasks respectively. Specifically, we compare the test time performance of FOMAML, MAML, pretrained agent, oracle and a random baseline. All agents were trained for the same number of iterations. These trained agents serve as the "intitializations" for the networks at test time, where the goal is to investigate how many gradient update iterations are needed from the new task to reach the full performance. We use the oracle reward numbers as quoted in the original MAML paper. We observe that the MAML and FOMAML agents are able to generalize very quickly to a new unseen task at test time (within 3 gradient updates), substantially outperforming pretrained and random-initialization baselines.

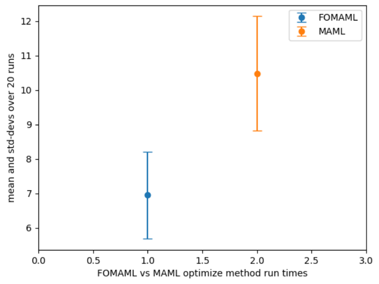

Figure 3: Comparison between FOMAML, MAML run times of main optimization loop(where meta-gradients are computed)

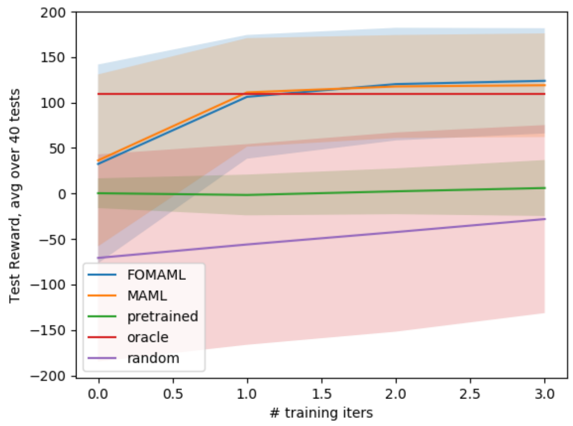

Figure 4: Comparison between FOMAML, MAML and other baselines for ant velocity task

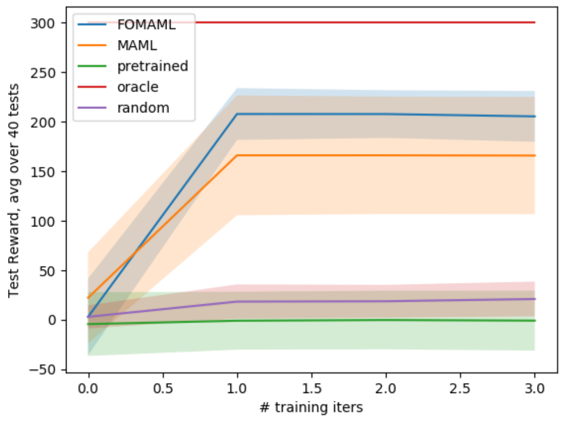

Figure 5: Comparison between FOMAML, MAML and other baselines for ant direction task

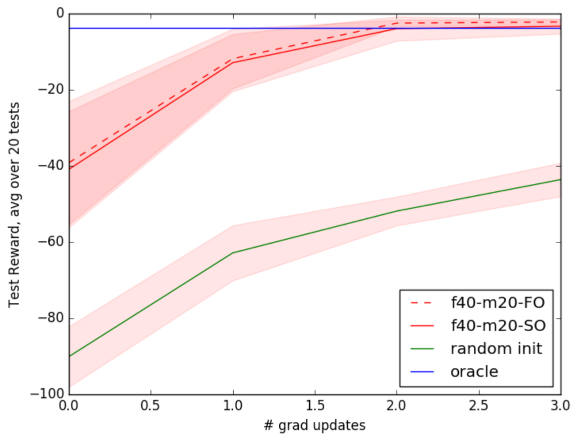

Figure 6: MAML and FOMAML compared to baselines for 2D Pointmass environment

## 5 Discussion and Future Work

Meta-learning has been of great interest recently, specially in the context of agents in dynamic environments and lifelong-learning [14].
In this work, we investigated the efficacy of the first-order approximation attempted to characterize the trade-off between accuracy and computational complexity of a meta reinforcement learning algorithm, using a first-derivative approximation. The experimental results suggest an approximate $1.5\times$ speedup while performing just as well as MAML on all three tasks.

In future work, we would like to better explain the mathematical underpinnings of the effect of the first-order approximation by quantifying the magnitude of the second-order derivative terms. While we have done this qualitatively by demonstrating similar performance on benchmark tasks, it would be interesting to see the actual derivative magnitudes that would allow us to quantify the magnitude of their effect on the parameter updates.

We would also like to test on other, more complex families of tasks in simulation.

## 6 Acknowledgements

We would like to express our deepest gratitude to Prof. Ruslan Salakhutdinov and the entire 10-703 team of instructors and TAs for guiding us during the project and the rest of the course!

## References

[1] Schmidt, L. A. (2009). Meaning and compositionality as statistical induction of categories and constraints (Doctoral dissertation, Massachusetts Institute of Technology).

[2] Nichol, A., and Schulman, J. (2018). Reptile: a Scalable Metalearning Algorithm. arXiv preprint arXiv:1803.02999.

[3] Parisotto, E., Ba, J.L. and Salakhutdinov, R., (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. arXiv preprint arXiv:1511.06342.

[4] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400.

[5] Schmidhuber, J. (1987) Evolutionary principles in self-referential learning. On learning how to learn: The meta-meta-... hook.) Diploma thesis, Institut f. Informatik, Tech. Univ. Munich

[6] Thrun, S., and Pratt, L. (1998). Learning to learn: Introduction and overview. In Learning to learn (pp. 3-17). Springer, Boston, MA.

[7] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. Science, 350(6266), 1332-1338

[8] Ravi, S. and Larochelle, H., (2016). Optimization as a model for few-shot learning.

[9] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T. and de Freitas, N., (2016). Learning to learn by gradient descent by gradient descent. In Advances in Neural Information Processing Systems (pp. 3981-3989).

[10] Li, K. and Malik, J., (2016). Learning to optimize. arXiv preprint arXiv:1606.01885.

[11] Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., (2015), June. Trust region policy optimization. In International Conference on Machine Learning (pp. 1889-1897).

[12] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E. and Darrell, T., (2014), January. Decaf: A deep convolutional activation feature for generic visual recognition. In International conference on machine learning (pp. 647-655).

[13] Goodfellow, I.J., Shlens, J. and Szegedy, C., (2014) Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

[14] Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., Abbeel, P. (2017). Continuous adaptation via meta-learning in nonstationary and competitive environments. arXiv preprint arXiv:1710.03641.

[15] Williams, R. J., Peng, J. (1989, June). Reinforcement learning algorithms as function optimizers. In Proceedings of the International Joint Conference on Neural Networks, Washington DC (Vol. 2, pp. 89-95).

[16] Todorov, E., Erez, T., Tassa, Y. (2012, October). Mujoco: A physics engine for model-based control. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on (pp. 5026-5033). IEEE.